Foreword

The *Handbook of Evolutionary Computation* represents a major milestone for the field of evolutionary computation (EC). As is the case with any new field, there are a number of distinct stages of growth and maturation. The field began in the late 1950s and early 1960s as the availability of digital computing permitted scientists and engineers to build and experiment with various models of evolutionary processes. This early work produced a number of important EC paradigms, including evolutionary programming (EP), evolution strategies (ESs), and genetic algorithms (GAs), which served as the basis for much of the work done in the 1970s, a period of intense exploration and refinement of these ideas. The result was a variety of robust algorithms with significant potential for addressing difficult scientific and engineering problems. By the late 1980s and early 1990s the level of activity had grown to the point that each of the subgroups associated with the primary EC paradigms (GAs, ESs, and EP) was involved in planning and holding its own regularly scheduled conferences.

However, within the field there was a growing sense of the need for more interaction and cohesion among the various subgroups. If the field as a whole were to mature, it needed a name, it needed to have an articulated cohesive structure, and it needed a reservoir for archival literature. The 1990s reflect this maturation with the choice of *evolutionary computation* as the name of the field, the establishment of two journals for the field, and the commitment to produce this handbook as the first clear and cohesive description of the field.

With the publication of this handbook there is now a sense of unity and maturity to the field. The handbook represents a momentous accomplishment for which we owe the editors and the many contributors a great deal of thanks. More importantly, it is designed to be an evolving description of the field and will continue to serve as a foundational reference for the future.

Kenneth De Jong, *George Mason University*
Lawrence Fogel, *Natural Selection Inc.*
Hans-Paul Schwefel, *University of Dortmund*

A1.1 Introduction

David B Fogel

Abstract

A rationale for simulating evolution is offered in this section. Efforts in evolutionary computation commonly derive from one of four different motivations: improving optimization, robust adaptation, machine intelligence, and facilitating a greater understanding of biology. A brief overview for each of these avenues is offered here.

A1.1.1 Introductory remarks

As a recognized field, *evolutionary computation* is quite young. The term itself was invented as recently as 1991, and it represents an effort to bring together researchers who have been following different approaches to simulating various aspects of evolution. These techniques of *genetic algorithms*, *evolution strategies*, and *evolutionary programming* have one fundamental commonality: they each involve the reproduction, random variation, competition, and selection of contending individuals in a population. These form the essential essence of evolution, and once these four processes are in place, whether in nature or in a computer, evolution is the inevitable outcome (Atmar 1994). The impetus to simulate evolution on a computer comes from at least four directions.

B1.2, B1.3 B1.4

A1.1.2 Optimization

Evolution is an optimization process (Mayr 1988, p 104). Darwin (1859, ch 6) was struck with the 'organs of extreme perfection' that have been evolved, one such example being the image-forming eye (Atmar, 1976). Optimization does not imply perfection, yet evolution can discover highly precise functional solutions to particular problems posed by an organism's environment, and even though the mechanisms that are evolved are often overly elaborate from an engineering perspective, function is the sole quality that is exposed to natural selection, and functionality is what is optimized by iterative selection and mutation.

It is quite natural, therefore, to seek to describe evolution in terms of an algorithm that can be used to solve difficult engineering optimization problems. The classic techniques of gradient descent, deterministic hill climbing, and purely random search (with no heredity) have been generally unsatisfactory when applied to nonlinear optimization problems, especially those with stochastic, temporal, or chaotic components. But these are the problems that nature has seemingly solved so very well. Evolution provides inspiration for computing the solutions to problems that have previously appeared intractable. This was a key foundation for the efforts in evolution strategies (Rechenberg 1965, 1994, Schwefel 1965, 1995).

A1.1.3 Robust adaptation

The real world is never static, and the problems of temporal optimization are some of the most challenging. They require changing behavioral strategies in light of the most recent feedback concerning the success or failure of the current strategy. Holland (1975), under the framework of genetic algorithms (formerly called *reproductive plans*), described a procedure that can evolve strategies, either in the form of coded strings or as explicit behavioral rule bases called *classifier systems*, by exploiting the potential to recombine successful pieces of competing strategies, bootstrapping the knowledge gained by independent individuals. The result is a robust procedure that has the potential to adjust performance based on feedback from the environment.

B1.5.2

A1.1.4 Machine intelligence

Intelligence may be defined as the capability of a system to adapt its behavior to meet desired goals in a range of environments (Fogel 1995, p xiii). Intelligent behavior then requires prediction, for adaptation to future circumstances requires predicting those circumstances and taking appropriate action. Evolution has created creatures of increasing intelligence over time. Rather than seek to generate machine intelligence by replicating humans, either in the rules they may follow or in their neural connections, an alternative approach to generating machine intelligence is to simulate evolution on a class of predictive algorithms. This was the foundation for the evolutionary programming research of Fogel (1962, Fogel *et al* 1966).

A1.1.5 Biology

Rather than attempt to use evolution as a tool to solve a particular engineering problem, there is a desire to capture the essence of evolution in a computer simulation and use the simulation to gain new insight into the physics of *natural evolutionary processes* (Ray 1991). Success raises the possibility of studying alternative biological systems that are merely plausible images of what life might be like in some way. It also raises the question of what properties such imagined systems might have in common with life as evolved on Earth (Langton 1987). Although every model is incomplete, and assessing what life might be like in other instantiations lies in the realm of pure speculation, computer simulations under the rubric of *artificial life* have generated some patterns that appear to correspond with naturally occurring phenomena.

A2.1

A1.1.6 Discussion

The ultimate answer to the question 'why simulate evolution?' lies in the lack of good alternatives. We cannot easily germinate another planet, wait several millions of years, and assess how life might develop elsewhere. We cannot easily use classic optimization methods to find global minima in functions when they are surrounded by local minima. We find that expert systems and other attempts to mimic human intelligence are often brittle: they are not robust to changes in the domain of application and are incapable of correctly predicting future circumstances so as to take appropriate action. In contrast, by successfully exploiting the use of randomness, or in others words *the useful use of uncertainty*, 'all possible pathways are open' for evolutionary computation (Hofstadter 1995, p 115). Our challenge is, at least in some important respects, to not allow our own biases to constrain the potential for evolutionary computation to discover new solutions to new problems in fascinating and unpredictable ways. However, as always, the ultimate advancement of the field will come from the careful abstraction and interpretation of the natural processes that inspire it.

References

Atmar J W 1976 *Speculation on the Evolution of Intelligence and its Possible Realization in Machine Form* Doctoral Dissertation, New Mexico State University

Atmar W 1994 Notes on the simulation of evolution *IEEE Trans. Neural Networks* **NN-5** 130–47

Darwin C R 1859 *On the Origin of Species by Means of Natural Selection or the Preservation of Favoured Races in the Struggle for Life* (London: Murray)

Fogel D B 1995 *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence* (Piscataway, NJ: IEEE)

Fogel L J 1962 Autonomous automata *Industr. Res.* **4** 14–9

Fogel L J, Owens A J and Walsh M J 1966 *Artificial Intelligence through Simulated Evolution* (New York: Wiley)

Hofstadter D 1995 *Fluid Concepts and Creative Analogies: Computer Models of the Fundamental Mechanisms of Thought* (New York: Basic Books)

Holland J H 1975 *Adaptation in Natural and Artificial Systems* (Ann Arbor, MI: University of Michigan Press)

Langton C G 1987 Artificial life *Artificial Life* ed C G Langton (Reading, MA: Addison-Wesley) pp 1–47

Mayr E 1988 *Toward a New Philosophy of Biology: Observations of an Evolutionist* (Cambridge, MA: Belknap)

Ray T 1991 An approach to the synthesis of life *Artificial Life II* ed C G Langton, C Taylor, J D Farmer and S Rasmussen (Reading, MA: Addison-Wesley) pp 371–408

Rechenberg I 1965 *Cybernetic Solution Path of an Experimental Problem* Royal Aircraft Establishment Library Translation 1122, Farnborough, UK

——1994 *Evolutionsstrategies '94* (Stuttgart: Frommann-Holzboog)

Schwefel H-P 1965 *Kybernetische Evolution als Strategie der Experimentellen Forschung in der Strömungstechnik* Diploma Thesis, Technical University of Berlin

——1995 *Evolution and Optimum Seeking* (New York: Wiley)

A1.2 Possible applications of evolutionary computation

David Beasley

Abstract

This section describes some of the applications to which evolutionary computation has been applied. Applications are divided into the areas of planning, design, simulation and identification, control, and classification.

A1.2.1 Introduction

Applications of evolutionary computation (EC) fall into a wide continuum of areas. For convenience, in this section they have been split into five broad categories:

- planning
- design
- simulation and identification
- control
- classification.

These categories are by no means meant to be absolute or definitive. They all overlap to some extent, and many applications could rightly appear in more than one of the categories.

These categories correspond to the sections found in Part F roughly as follows: *planning* is covered by F1.5 (scheduling) and F1.7 (packing); *simulation* and *identification* are covered by F1.4 (identification), F1.8 (simulation models) and F1.10 (simulated evolution); *control* is covered by F1.3 (control); *classification* is covered by F1.6 (pattern recognition). *Design* is not covered by a specific section in Part F.

F1.5, F1.7, F1.4
F1.8, F1.10
F1.3, F1.6

Some of the applications mentioned here are described more fully in other parts of this book as indicated by marginal cross-references. The final part of this section lists a number of bibliographies where more extensive information on EC applications can be found.

A1.2.2 Applications in planning

A1.2.2.1 Routing

Perhaps one of the best known combinatorial optimization problems is the *traveling salesman problem* or TSP (Goldberg and Lingle 1985, Grefenstette 1987, Fogel 1988, Oliver *et al* 1987, Mühlenbein 1989, Whitley *et al* 1989, Fogel 1993a, Homaifar *et al* 1993). A salesman must visit a number of cities, and then return home. In which order should the cities be visited to minimize the distance traveled? Optimizing the tradeoff between speed and accuracy of solution has been one aim (Verhoeven *et al* 1992).

G1.3, G9.5

A generalization of the TSP occurs when there is more than one salesman (Fogel 1990). The *vehicle routing problem* is similar. There is a fleet of vehicles, all based at the same depot. A set of customers must each receive one delivery. Which route should each vehicle take for minimum cost? There are constraints, for example, on vehicle capacity and delivery times (Blanton and Wainwright 1993, Thangia *et al* 1993).

Closely related to this is the *transportation problem*, in which a single commodity must be distributed

G9.8

to a number of customers from a number of depots. Each customer may receive deliveries from one or more depots. What is the minimum-cost solution? (Michalewicz 1992, 1993).

Planning the path which a *robot* should take is another route planning problem. The path must be feasible and safe (i.e. it must be achievable within the operational constraints of the robot) and there must be no collisions. Examples include determining the joint motions required to move the gripper of a robot arm between locations (Parker *et al* 1989, Davidor 1991, McDonnell *et al* 1992), and autonomous vehicle routing (Jakob *et al* 1992, Page *et al* 1992). In unknown areas or nonstatic environments, on-line *planning/navigating* is required, in which the robot revises its plans as it travels. G3.6

A1.2.2.2 Scheduling

Scheduling involves devising a plan to carry out a number of activities over a period of time, where the activities require resources which are limited, there are various constraints and there are one or more objectives to be optimized.

Job shop scheduling is a widely studied NP-complete problem (Davis 1985, Biegel and Davern 1990, G1.2.3 Syswerda 1991, Yamada and Nakano 1992). The scenario is a manufacturing plant, with machines of different types. There are a number of jobs to be completed, each comprising a set of tasks. Each task requires a particular type of machine for a particular length of time, and the tasks for each job must be completed in a given order. What schedule allows all tasks to be completed with minimum cost? Husbands (1993) has used the additional biological metaphor of an *ecosystem*. His method optimizes the sequence G9.3 of tasks in each job at the same time as it builds the schedule. In real job shops the requirements may change while the jobs are being carried out, requiring that the schedule be replanned (Fang *et al* 1993). In the limit, the manufacturing process runs continuously, so all scheduling must be carried out on-line, as in a chemical flowshop (Cartwright and Tuson 1994).

Another scheduling problem is to devise a timetable for a set of examinations (Corne *et al* 1994), university lectures (Ling 1992), a *staff rota* (Easton and Mansour 1993) or suchlike. G9.4

In computing, scheduling problems include efficiently allocating tasks to processors in a multiprocessor system (Van Driessche and Piessens 1992, Kidwell 1993, Fogel and Fogel 1996), and devising memory cache replacement policies (Altman *et al* 1993).

A1.2.2.3 Packing

Evolutionary algorithms (EAs) have been applied to many packing problems, the simplest of which is the one-dimensional *zero–one knapsack problem*. Given a knapsack of a certain capacity, and a set of G9.7 items, each with a particular size and value, find the set of items with maximum value which can be accommodated in the knapsack. Various real-world problems are of this type: for example, the allocation of communication channels to customers who are charged at different rates.

There are various examples of two-dimensional packing problems. When manufacturing items are cut from sheet materials (e.g. metal or cloth), it is desirable to find the most compact arrangement of pieces, so as to minimize the amount of scrap (Smith 1985, Fujita *et al* 1993). A similar problem arises in the design of layouts for integrated circuits—how should the subcircuits be arranged to minimize the total chip area required (Fourman 1985, Cohoon and Paris 1987, Chan *et al* 1991)?

In three dimensions, there are obvious applications in which the best way of packing objects into a restricted space is required. Juliff (1993) has considered the problem of packing goods into a truck for delivery. (See also Section F1.7 of this handbook.) F1.7

A1.2.3 Applications in design

The design of *filters* has received considerable attention. EAs have been used to design electronic or digital G3.1 systems which implement a desired frequency response. Both finite impulse response (FIR) and infinite impulse response (IIR) filter structures have been employed (Etter *et al* 1982, Suckley 1991, Fogel 1991, Fonseca *et al* 1993, Ifeachor and Harris 1993, Namibar and Mars 1993, Roberts and Wade 1993, Schaffer and Eshelman 1993, White and Flockton 1993, Wicks and Lawson 1993, Wilson and Macleod 1993). EAs have also been used to optimize the design of signal processing systems (San Martin and Knight 1993) and in integrated circuit design (Louis and Rawlins 1991, Rahmani and Ono 1993). The *unequal-area facility layout problem* (Smith and Tate 1993) is similar to integrated circuit design. It involves finding

a two-dimensional arrangement of 'departments' such that the distance which information has to travel between departments is minimized.

EC techniques have been widely applied to *artificial neural networks*, both in the design of network topologies and in the search for optimum sets of weights (Miller *et al* 1989, Fogel *et al* 1990, Harp and Samad 1991, Baba 1992, Hancock 1992, Feldman 1993, Gruau 1993, Polani and Uthmann 1993, Romaniuk 1993, Spittle and Horrocks 1993, Zhang and Mühlenbein 1993, Porto *et al* 1995). They have also been applied to Kohonen feature map design (Polani and Uthmann 1992). Other types of network design problems have also been approached (see Section G1.3 of this handbook), for example, in telecommunications (Cox *et al* 1991, Davis and Cox 1993). G1.3

There have been many *engineering* applications of EC: structure design, both two-dimensional, such as a plane truss (Lohmann 1992, Watabe and Okino 1993), and three-dimensional, such as aircraft design (Bramlette and Bouchard 1991), actuator placement on space structures (Furuya and Haftka 1993), *linear accelerator* design, gearbox design, and chemical reactor design (Powell and Skolnick 1993). In relation to high-energy physics, the design of *Monte Carlo generators* has been tackled. G3 G4.2 G4.1

In order to perform parallel computations requiring global coordination, EC has been used to design *cellular automata* with appropriate communication mechanisms. G1.6

There have also been applications in *testing and fault diagnosis*. For example, an EA can be used to search for challenging fault scenarios for an *autonomous vehicle controller*. G3.4

A1.2.4 Applications in simulation and identification

Simulation involves taking a design or model for a system, and determining how the system will behave. In some cases this is done because we are unsure about the behavior (e.g. when designing a new aircraft). In other cases, the behavior is known, but we wish to test the accuracy of the model (e.g. see Section F1.8 of this handbook). F1.8

EC has been applied to difficult problems in *chemistry* and *biology*. Roosen and Meyer (1992) used an evolution strategy to determine the equilibrium of chemically reactive systems, by determining the minimum free enthalpy of the compounds involved. The determination of the three-dimensional structure of a protein, given its amino acid sequence, has been tackled (Lucasius *et al* 1991). Lucasius and Kateman (1992) approached this as a sequenced subset selection problem, using two-dimensional nuclear magnetic resonance spectrum data as a starting point. Others have searched for energetically favorable protein conformations (Schulze-Kremer 1992, Unger and Moult 1993), and used EC to assist with drug design (Gehlhaar *et al* 1995). EC has been used to simulate how the nervous system learns in order to test an existing theory (see Section G8.4 of this handbook). Similarly, EC has been used in order to help develop models of biological evolution (see Section F1.10 of this handbook). G5, G6 G8.4 F1.10

In the field of *economics*, EC has been used to model *economic interaction* of competing firms in a market. G7.1

Identification is the inverse of simulation. It involves determining the design of a system given its behavior. F1.4

Many systems can be represented by a model which produces a single-valued output in response to one or more input signals. Given a number of observations of input and output values, *system identification* is the task of deducing the details of the model. Flockton and White (1993) concern themselves with determining the poles and zeros of the system. G1.4

One reason for wanting to identify systems is so that we can predict the output in response to a given set of inputs. In Section G4.3 of this handbook EC is employed to fit equations to noisy, chaotic medical data, in order to predict future values. Janikow and Cai (1992) similarly used EC to estimate statistical functions for survival analysis in clinical trials. In a similar area, Manela *et al* (1993) used EC to fit spline functions to noisy pharmaceutical fermentation process data. G4.3

In Section G5.1 of this handbook, EC is used to identify the sources of airborne pollution, given data from a number of monitoring points in an urban area—the *source apportionment problem*. G5.1

In *electromagnetics*, Tanaka *et al* (1993) have applied EC to determining the two-dimensional current distribution in a conductor, given its external magnetic field.

Away from conventional *system* identification, in Section G8.3 of this handbook, an EC approach was used to help with identifying criminal suspects. This system helps witnesses to create a likeness of the suspect, without the need to give an explicit description. G8.3

A1.2.5 Applications in control

There are two distinct approaches to the use of EC in control: *off-line* and *on-line*. The off-line approach uses an EA to design a controller, which is then used to control the system. The on-line approach uses an EA as an active part of the control process. Therefore, with the off-line approach there is nothing evolutionary about the control process itself, only about the design of the controller.

Some researchers (Fogel *et al* 1966, DeJong 1980) have sought to use the adaptive qualities of EAs in order to build on-line controllers for dynamic systems. The advantage of an evolutionary controller is that it can adapt to cope with systems whose characteristics change over time, whether the change is gradual or sudden. Most researchers, however, have taken the off-line approach to the control of relatively unchanging systems.

Fonseca and Fleming (1993) used an EA to design a controller for a gas turbine engine, to optimize its step response. A control system to optimize combustion in multiple-burner furnaces and boiler plants is discussed in Section G3.2. EC has also been applied to the control of guidance and navigation systems G3.2 (Krishnakumar and Goldberg 1990, 1992).

Hunt (1992b) has tackled the problem of synthesizing LQG (linear–quadratic–Gaussian) and H_∞ (H-infinity) optimal controllers. He has also considered the frequency domain optimization of controllers with fixed structures (Hunt 1992a).

Two control problems which have been well studied are balancing a pole on a movable cart (Fogel 1995), and backing up a trailer truck to a loading bay from an arbitrary starting point (Abu Zitar and Hassoun 1993). In robotics, EAs have been developed which can evolve control systems for visually guided behaviors (see Section G3.7). They can also learn how to control mobile robots (Kim and Shim G3.7 1995), for example, controlling the legs of a six-legged 'insect' to make it crawl or walk (Spencer 1993). Almássy and Verschure (1992) modeled the interaction between natural selection and the adaptation of individuals during their lifetimes to develop an agent with a distributed adaptive control framework which learns to avoid obstacles and locate food sources.

A1.2.6 Applications in classification

A significant amount of EC research has concerned the theory and practice of *classifier systems* (CFS) B1.5.2 (Booker 1985, Holland 1985, 1987, Holland *et al* 1987, Robertson 1987, Wilson 1987, Fogarty 1994). Classifier systems are at the heart of many other types of system. For example, many control systems rely on being able to *classify* the characteristics of their environment before an appropriate control decision can be made. This is true in many *robotics* applications of EC, for example, learning to control robot arm motion (Patel and Dorigo 1994) and learning to solve mazes (Pipe and Carse 1994).

An important aspect of a classifier system, especially in a control application, is how the state space is partitioned. Many applications take for granted a particular partitioning of the state space, while in others, the appropriate partitioning of the state space is itself part of the problem (Melhuish and Fogarty 1994). In Section G2.3, EC was used to determine optimal symbolic descriptions for concepts. G2.3

Game playing is another application for which classification plays a key role. Although EC is often applied to rather simple games (e.g. the prisoner's dilemma (Axelrod 1987, Fogel 1993b)), this is sometimes motivated by more serious applications, such as military ones (e.g. the two-tanks game (Fairley and Yates 1994) and air *combat maneuvering*. G3.3

EC has been hybridized with feature partitioning and applied to a range of tasks (Güvenir and Şirin 1993), including classification of iris flowers, prediction of survival for heart attack victims from echocardiogram data, diagnosis of heart disease, and classification of glass samples. In *linguistics*, EC has been applied to the classification of Swedish words.

In *economics*, Oliver (1993) has found rules to reflect the way in which consumers choose one brand rather than another, when there are multiple criteria on which to judge a product. A fuzzy hybrid system has been used for *financial decision making*, with applications to credit evaluation, risk assessment, and G7.2 insurance underwriting.

In *biology*, EC has been applied to the difficult task of protein secondary-structure determination, for example, classifying the locations of particular *protein segments* (Handley 1993). It has also been applied G6.1 to the classification of soil samples (Punch *et al* 1993).

In *image processing*, there have been further military applications, classifying features in images as targets (Bala and Wechsler 1993, Tackett 1993), and also non-military applications, such as *optical character recognition*. G8.1

Of increasing importance is the efficient storage and retrieval of *information*. Section G2.2 is concerned with generating equifrequency distributions of material, to improve the efficiency of information storage and its subsequent retrieval. EC has also been employed to assist with the representation and storage of chemical structures, and the retrieval from databases of molecules containing certain substructures (Jones *et al* 1993). The retrieval of *documents* which match certain characteristics is becoming increasingly important as more and more information is held on-line. Tools to retrieve documents which contain specified words have been available for many years, but they have the limitation that constructing an appropriate search query can be difficult. Researchers are now using EAs to help with *query construction* (Yang and Korfhage 1993). G2.2 G2.1

A1.2.7 Summary

EC has been applied in a vast number of application areas. In some cases it has advantages over existing computerized techniques. More interestingly, perhaps, it is being applied to an increasing number of areas in which computers have not been used before. We can expect to see the number of applications grow considerably in the future. Comprehensive bibliographies in many different application areas are listed in the further reading section of this article.

References

Abu Zitar R A and Hassoun M H 1993 Regulator control via genetic search and assisted reinforcement *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 254–62

Almássy N and Verschure P 1992 Optimizing self-organising control architectures with genetic algorithms: the interaction between natural selection and ontogenesis *Parallel Problem Solving from Nature, 2 (Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature, Brussels, 1992)* ed R Männer and B Manderick (Amsterdam: Elsevier) pp 451–60

Altman E R, Agarwal V K and Gao G R 1993 A novel methodology using genetic algorithms for the design of caches and cache replacement policy *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 392–9

Axelrod R 1987 The evolution of strategies in the iterated prisoner's dilemma *Genetic Algorithms and Simulated Annealing* ed L Davis (Boston, MA: Pitman) ch 3, pp 32–41

Baba N 1992 Utilization of stochastic automata and genetic algorithms for neural network learning *Parallel Problem Solving from Nature, 2 (Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature, Brussels, 1992)* ed R Männer and B Manderick (Amsterdam: Elsevier) pp 431–40

Bagchi S, Uckun S, Miyabe Y and Kawamura K 1991 Exploring problem-specific recombination operators for job shop scheduling *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, July 1991)* ed R Belew and L Booker (San Mateo, CA: Morgan Kaufmann) pp 10–7

Bala J W and Wechsler H 1993 Learning to detect targets using scale-space and genetic search *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 516–22

Biegel J E and Davern J J 1990 Genetic algorithms and job shop scheduling *Comput. Indust. Eng.* **19** 81–91

Blanton J L and Wainwright R L 1993 Multiple vehicle routing with time and capacity constraints *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 452–9

Booker L 1985 Improving the performance of genetic algorithms in classifier systems *Proc. 1st Int. Conf. on Genetic Algorithms (Pittsburgh, PA, July 1985)* ed J J Grefenstette (Hillsdale, NJ: Lawrence Erlbaum Associates) pp 80–92

Bramlette M F and Bouchard E E 1991 Genetic algorithms in parametric design of aircraft *Handbook of Genetic Algorithms* ed L Davis (New York: Van Nostrand Reinhold) ch 10, pp 109–23

Cartwright H M and Tuson A L 1994 Genetic algorithms and flowshop scheduling: towards the development of a real-time process control system *Evolutionary Computing (AISB Workshop, Leeds, 1994, Selected Papers) (Lecture Notes in Computer Science 865)* ed T C Fogarty (Berlin: Springer) pp 277–90

Chan H, Mazumder P and Shahookar K 1991 Macro-cell and module placement by genetic adaptive search with bitmap-represented chromosome *Integration VLSI J.* **12** 49–77

Cohoon J P and Paris W D 1987 Genetic placement *IEEE Trans. Computer-Aided Design* **CAD-6** 956–64

Corne D, Ross P and Fang H-L 1994 Fast practical evolutionary timetabling *Evolutionary Computing (AISB Workshop, Leeds, 1994, Selected Papers) (Lecture Notes in Computer Science 865)* ed T C Fogarty (Berlin: Springer) pp 250–63

Cox L A, Davis L and Qiu Y 1991 Dynamic anticipatory routing in circuit-switched telecommunications networks *Handbook of Genetic Algorithms* ed L Davis (New York: Van Nostrand Reinhold) ch 11, pp 124–43

Davidor Y 1991 A genetic algorithm applied to robot trajectory generation *Handbook of Genetic Algorithms* ed L Davis (New York: Van Nostrand Reinhold) ch 12, pp 144–65

Davis L 1985 Job shop scheduling with genetic algorithms *Proc. 1st Int. Conf. on Genetic Algorithms (Pittsburgh, PA, July 1985)* ed J J Grefenstette (Hillsdale, NJ: Lawrence Erlbaum Associates) pp 136–40

Davis L and Cox A 1993 A genetic algorithm for survivable network design *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 408–15

DeJong K 1980 Adaptive system design: a genetic approach *IEEE Trans. Systems, Man Cybern.* **SMC-10** 566–74

Easton F F and Mansour N 1993 A distributed genetic algorithm for employee staffing and scheduling problems *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 360–67

Etter D M, Hicks M J and Cho K H 1982 Recursive adaptive filter design using an adaptive genetic algorithm *IEEE Int. Conf. on Acoutics, Speech and Signal Processing* (Piscataway, NJ: IEEE) pp 635–8

Fairley A and Yates D F 1994 Inductive operators and rule repair in a hybrid genetic learning system: some initial results *Evolutionary Computing (AISB Workshop, Leeds, 1994, Selected Papers) (Lecture Notes in Computer Science 865)* ed T C Fogarty (Berlin: Springer) pp 166–79

Fang H-L, Ross P and Corne D 1993 A promising genetic algorithm approach to job-shop scheduling, rescheduling and open-shop scheduling problems *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 375–82

Feldman D S 1993 Fuzzy network synthesis with genetic algorithms *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 312–7

Flockton S J and White M 1993 Pole-zero system identification using genetic algorithms *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 531–5

Fogarty T C 1994 Co-evolving co-operative populations of rules in learning control systems *Evolutionary Computing (AISB Workshop, Leeds, 1994, Selected Papers) (Lecture Notes in Computer Science 865)* ed T C Fogarty (Berlin: Springer) pp 195–209

Fogel D B 1988 An evolutionary approach to the traveling salesman problem *Biol. Cybernet.* **6** 139–44

——1990 A parallel processing approach to a multiple traveling salesman problem using evolutionary programming *Proc. 4th Ann. Symp. on Parallel Processing* (Piscataway, NJ: IEEE) pp 318–26

——1991 *System Identification through Simulated Evolution* (Needham, MA: Ginn)

——1993a Applying evolutionary programming to selected traveling salesman problems *Cybernet. Syst.* **24** 27–36

——1993b Evolving behaviors in the iterated prisoner's dilemma *Evolut. Comput.* **1** 77–97

——1995 *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence* (Piscataway, NJ: IEEE)

Fogel D B and Fogel L J 1996 Using evolutionary programming to schedule tasks on a suite of heterogeneous computers *Comput. Operat. Res.* **23** 527–34

Fogel D B, Fogel L J and Porto V W 1990 Evolving neural networks *Biol. Cybern.* **63** 487–93

Fogel L J, Owens A J and Walsh M J 1966 *Artificial intelligence Through Simulated Evolution* (New York: Wiley)

Fonseca C M and Fleming P J 1993 Genetic algorithms for multiobjective optimization: formulation, discussion and generalization *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 416–23

Fonseca C M, Mendes E M, Fleming P J and Billings S A 1993 Non-linear model term selection with genetic algorithms *Natural Algorithms in Signal Processing (Workshop, Chelmsford, UK, November 1993)* vol 2 (London: IEE) pp 27/1–27/8

Fourman M P 1985 Compaction of symbolic layout using genetic algorithms *Proc. 1st Int. Conf. on Genetic Algorithms (Pittsburgh, PA, July 1985)* ed J J Grefenstette (Hillsdale, NJ: Lawrence Erlbaum Associates) pp 141–53

Fujita K, Akagi S and Hirokawa N 1993 Hybrid approach for optimal nesting using genetic algorithm and a local minimization algorithm *Advances in Design Automation* vol 1, DE-65–1 (ASME) pp 477–84

Furuya H and Haftka R T 1993 Genetic algorithms for placing actuators on space structures *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 536–42

Gehlhaar D K, Verkhivker G M, Rejto P A, Sherman C J, Fogel D B, Fogel L J and Freer S T 1995 Molecular recognition of the inhibitor AG-1343 by HIV-1 protease: conformationally flexible docking by evolutionary programming *Chem. Biol.* **2** 317–24

Goldberg D E and Lingle R 1985 Alleles, loci and the travelling salesman problem *Proc. 1st Int. Conf. on Genetic Algorithms (Pittsburgh, PA, July 1985)* ed J J Grefenstette (Hillsdale, NJ: Lawrence Erlbaum Associates) pp 154–9

Grefenstette J J 1987 Incorporating problem specific knowledge into genetic algorithms *Genetic Algorithms and Simulated Annealing* ed L Davis (Boston, MA: Pitman) ch 4, pp 42–60

Gruau F 1993 Genetic synthesis of modular neural networks *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 318–25

Güvenir H A and Şirin I 1993 A genetic algorithm for classification by feature recognition *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 543–8

Hancock P 1992 Recombination operators for the design of neural nets by genetic algorithm *Parallel Problem Solving from Nature, 2 (Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature, Brussels, 1992)* ed R Männer and B Manderick (Amsterdam: Elsevier) pp 441–50

Handley S 1993 Automated learning of a detector for α-helices in protein sequences via genetic programming *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 271–8

Harp S A and Samad T 1991 Genetic synthesis of neural network architecture *Handbook of Genetic Algorithms* ed L Davis (New York: Van Nostrand Reinhold) ch 15, pp 202–21

Holland J H 1985 Properties of the bucket-brigade algorithm *Proc. 1st Int. Conf. on Genetic Algorithms (Pittsburgh, PA, July 1985)* ed J J Grefenstette (Hillsdale, NJ: Lawrence Erlbaum Associates) pp 1–7

——1987 Genetic algorithms and classifier systems: foundations and future directions *Proc. 2nd Int. Conf. on Genetic Algorithms (Pittsburgh, PA, July 1985)* ed J J Grefenstette (Hillsdale, NJ: Lawrence Erlbaum Associates) pp 82–9

Holland J H, Holyoak K J, Nisbett R E and Thagard P R 1987 Classifier systems, Q-morphisms and induction *Genetic Algorithms and Simulated Annealing* ed L Davis (Boston, MA: Pitman) ch 9, pp 116–28

Homaifar, A., Guan S and Liepins G 1993 A new approach to the travelling salesman problem by genetic algorithms *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 460–6

Hunt K J 1992a Optimal control system synthesis with genetic algorithms *Parallel Problem Solving from Nature, 2 (Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature, Brussels, 1992)* ed R Männer and B Manderick (Amsterdam: Elsevier) pp 381–9

——1992b Polynomial LQG and H_∞ controller synthesis: a genetic algorithm solution *Proc. IEEE Conf. on Decision and Control (Tuscon, AZ)* (Picataway, NJ: IEEE)

Husbands P 1993 An ecosystems model for integrated production planning *Int. J. Comput. Integrated Manufacturing* **6** 74–86

Ifeachor E C and Harris S P 1993 A new approach to frequency sampling filter design using genetic algorithms *Natural Algorithms in Signal Processing (Workshop, Chelmsford, UK, November 1993)* vol 1 (London: IEE) pp 5/1–5/8

Jakob W, Gorges-Schleuter M and Blume C 1992 Application of genetic algorithms to task planning and learning *Parallel Problem Solving from Nature, 2 (Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature, Brussels, 1992)* ed R Männer and B Manderick (Amsterdam: Elsevier) pp 291–300

Janikow C Z and Cai H 1992 A genetic algorithm application in nonparametric functional estimation *Parallel Problem Solving from Nature, 2 (Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature, Brussels, 1992)* ed R Männer and B Manderick (Amsterdam: Elsevier) pp 249–58

Jones G, Brown R D, Clark D E, Willett P and Glen R C 1993 Searching databases of two-dimensional and three dimensional chemical structures using genetic algorithms *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 597–602

Juliff K 1993 A multi-chromosome genetic algorithm for pallet loading *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 467–73

Kidwell M D 1993 Using genetic algorithms to schedule distributed tasks on a bus-based system *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 368–74

Kim J-H and Shim H-S 1995 Evolutionary programming-based optimal robust locomotion control of autonomous mobile robots *Proc. 4th Ann. Conf. on Evolutionary Programming* ed J R McDonnell, R G Reynolds and D B Fogel (Cambridge, MA: MIT Press) pp 631–44

Krishnakumar K and Goldberg D E 1990 Genetic algorithms in control system optimization. *Proc. AIAA Guidance, Navigation, and Control Conf. (Portland, OR)* pp 1568–77

——1992 Control system optimization using genetic algorithms *J. Guidance Control Dynam.* **15** 735–40

Ling S-E 1992 Integrating genetic algorithms with a prolog assignment program as a hybrid solution for a polytechnic timetable problem *Parallel Problem Solving from Nature, 2 (Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature, Brussels, 1992)* ed R Männer and B Manderick (Amsterdam: Elsevier) pp 321–9

Lohmann R 1992 Structure evolution and incomplete induction *Parallel Problem Solving from Nature, 2 (Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature, Brussels, 1992)* ed R Männer and B Manderick (Amsterdam: Elsevier) pp 175–85

Louis S J and Rawlins G J E 1991 Designer genetic algorithms: genetic algorithms in structure design *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, July 1991)* ed R Belew and L Booker (San Mateo, CA: Morgan Kaufmann) pp 53–60

Lucasius C B, Blommers M J, Buydens L M and Kateman G 1991 A genetic algorithm for conformational analysis of DNA *Handbook of Genetic Algorithms* ed L Davis (New York: Van Nostrand Reinhold) ch 18, pp 251–81

Lucasius C B and Kateman G 1992 Towards solving subset selection problems with the aid of the genetic algorithm *Parallel Problem Solving from Nature, 2 (Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature, Brussels, 1992)* ed R Männer and B Manderick (Amsterdam: Elsevier) pp 239–47

Manela, M., Thornhill N and Campbell J A 1993 Fitting spline functions to noisy data using a genetic algorithm *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 549–56

McDonnell J R, Andersen B L, Page W C and Pin F G 1992 Mobile manipulator configuration optimization using evolutionary programming *Proc. 1st Ann. Conf. on Evolutionary Programming* ed D B Fogel and W Atmar (La Jolla, CA: Evolutionary Programming Society) pp 52–62

Melhuish C and Fogarty T C 1994 Applying a restricted mating policy to determine state space niches using immediate and delayed reinforcement *Evolutionary Computing (AISB Workshop, Leeds, 1994, Selected Papers) (Lecture Notes in Computer Science 865)* ed T C Fogarty (Berlin: Springer) pp 224–37

Michalewicz Z 1992 *Genetic Algorithms + Data Structures = Evolution Programs* (Berlin: Springer)

——1993 A hierarchy of evolution programs: an experimental study *Evolut. Comput.* **1** 51–76

Miller G F, Todd P M and Hegde S U 1989 Designing neural networks using genetic algorithms. *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, June 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 379–84

Mühlenbein H 1989 Parallel genetic algorithms, population genetics and combinatorial optimization *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, June 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 416–21

Namibar R and Mars P 1993 Adaptive IIR filtering using natural algorithms *Natural Algorithms in Signal Processing (Workshop, Chelmsford, UK, November 1993)* vol 2 (London: IEE) pp 20/1–20/10

Oliver J R 1993 Discovering individual decision rules: an application of genetic algorithms *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 216–22

Oliver I M, Smith D J and Holland J R C 1987 A study of permutation crossover operators on the travelling salesman problem *Proc. 2nd Int. Conf. on Genetic Algorithms (Cambridge, MA, 1987)* ed J J Grefenstette (Hillsdale, NJ: Erlbaum) pp 224–30

Page W C, McDonnell J R and Anderson B 1992 An evolutionary programming approach to multi-dimensional path planning *Proc. 1st Ann. Conf. on Evolutionary Programming* ed D B Fogel and W Atmar (La Jolla, CA: Evolutionary Programming Society) pp 63–70

Parker J K, Goldberg D E and Khoogar A R 1989 Inverse kinematics of redundant robots using genetic algorithms *Proc. Int. Conf. on Robotics and Automation (Scottsdale, AZ, 1989)* vol 1 (Los Alamitos: IEEE Computer Society Press) pp 271–6

Patel M J and Dorigo M 1994 Adaptive learning of a robot arm *Evolutionary Computing (AISB Workshop, Leeds, 1994, Selected Papers) (Lecture Notes in Computer Science 865)* ed T C Fogarty (Berlin: Springer) pp 180–94

Pipe A G and Carse B 1994 A comparison between two architectures for searching and learning in maze problems *Evolutionary Computing (AISB Workshop, Leeds, 1994, Selected Papers) (Lecture Notes in Computer Science 865)* ed T C Fogarty (Berlin: Springer) pp 238–49

Polani D and Uthmann T 1992 Adaptation of Kohonen feature map topologies by genetic algorithms *Parallel Problem Solving from Nature, 2 (Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature, Brussels, 1992)* ed R Männer and B Manderick (Amsterdam: Elsevier) pp 421–9

——1993 Training Kohonen feature maps in different topologies: an analysis using genetic algorithms *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 326–33

Porto V W, Fogel D B and Fogel L J 1995 Alternative neural network training methods *IEEE Expert* **10** 16–22

Powell D and Skolnick M M 1993 Using genetic algorithms in engineering design optimization with non-linear constraints *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 424–31

Punch W F, Goodman E D, Pei, M, Chia-Shun L, Hovland P and Enbody R 1993 Further research on feature selection and classification using genetic algorithms *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 557–64

Rahmani A T and Ono N 1993 A genetic algorithm for channel routing problem *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 494–8

Roberts A and Wade G 1993 A structured GA for FIR filter design *Natural Algorithms in Signal Processing (Workshop, Chelmsford, UK, November 1993)* vol 1 (London: IEE) pp 16/1–16/8

Robertson G 1987 Parallel implementation of genetic algorithms in a classifier system *Genetic Algorithms and Simulated Annealing* ed L Davis (Boston, MA: Pitman) ch 10, pp 129–40

Romaniuk S G 1993 Evolutionary growth perceptrons *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 334–41

Roosen P and Meyer F 1992 Determination of chemical equilibria by means of an evolution strategy *Parallel Problem Solving from Nature, 2 (Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature, Brussels, 1992)* ed R Männer and B Manderick (Amsterdam: Elsevier) pp 411–20

San Martin R and Knight J P 1993 Genetic algorithms for optimization of integrated circuit synthesis *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 432–8

Schaffer J D and Eshelman L J 1993 Designing multiplierless digital filters using genetic algorithms *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 439–44

Schulze-Kremer S 1992 Genetic algorithms for protein tertiary structure prediction *Parallel Problem Solving from Nature, 2 (Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature, Brussels, 1992)* ed R Männer and B Manderick (Amsterdam: Elsevier) pp 391–400

Smith A E and Tate D M 1993 Genetic optimization using a penalty function *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 499–505

Smith D 1985 Bin packing with adaptive search *Proc. 1st Int. Conf. on Genetic Algorithms (Pittsburgh, PA, July 1985)* ed J J Grefenstette (Hillsdale, NJ: Lawrence Erlbaum Associates)

Spencer G F 1993 Automatic generation of programs for crawling and walking *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) p 654

Spittle M C and Horrocks D H 1993 Genetic algorithms and reduced complexity artificial neural networks *Natural Algorithms in Signal Processing (Workshop, Chelmsford, UK, November 1993)* vol 1 (London: IEE) pp 8/1–8/9

Suckley D 1991 Genetic algorithm in the design of FIR filters *IEE Proc.* G **138** 234–8

Syswerda G 1991 Schedule optimization using genetic algorithms *Handbook of Genetic Algorithms* ed L Davis(New York: Van Nostrand Reinhold) ch 21, pp 332–49

Tackett W A 1993 Genetic programming for feature discovery and image discrimination *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 303–9

Tanaka, Y., Ishiguro A and Uchikawa Y 1993 A genetic algorithms application to inverse problems in electromagnetics *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) p 656

Thangia S R, Vinayagamoorthy R and Gubbi A V 1993 Vehicle routing with time deadlines using genetic and local algorithms *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 506–13

Unger R and Moult J 1993 A genetic algorithm for 3D protein folding simulations *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 581–8

Van Driessche R and Piessens R 1992 Load balancing with genetic algorithms *Parallel Problem Solving from Nature, 2 (Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature, Brussels, 1992)* ed R Männer and B Manderick (Amsterdam: Elsevier) pp 341–50

Verhoeven M G A, Aarts E H L, van de Sluis E and Vaessens R J M 1992 Parallel local search and the travelling salesman problem *Parallel Problem Solving from Nature, 2 (Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature, Brussels, 1992)* ed R Männer and B Manderick (Amsterdam: Elsevier) pp 543–52

Watabe H and Okino N 1993 A study on genetic shape design *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 445–50

White M and Flockton S 1993 A comparative study of natural algorithms for adaptive IIR filtering *Natural Algorithms in Signal Processing (Workshop, Chelmsford, UK, November 1993)* vol 2 (London: IEE) pp 22/1–22/8

Whitley D, Starkweather T and Fuquay D 1989 Scheduling problems and travelling salesmen: the genetic edge recombination operator *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, June 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 133–40

Wicks T and Lawson S 1993 Genetic algorithm design of wave digital filters with a restricted coefficient set *Natural Algorithms in Signal Processing (Workshop, Chelmsford, UK, November 1993)* vol 1 (London: IEE) pp 17/1–17/7

Wilson P B and Macleod M D 1993 Low implementation cost IIR digital filter design using genetic algorithms *Natural Algorithms in Signal Processing (Workshop, Chelmsford, UK, November 1993)* vol 1 (London: IEE) pp 4/1–4/8

Wilson S W 1987 Hierarchical credit allocation in a classifier system *Genetic Algorithms and Simulated Annealing* ed L Davis (Boston, MA: Pitman) ch 8, pp 104–15

Yamada T and Nakano R 1992 A genetic algorithm applicable to large-scale job-shop problems *Parallel Problem Solving from Nature, 2 (Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature, Brussels, 1992)* ed R Männer and B Manderick (Amsterdam: Elsevier) pp 281–90

Yang J-J and Korfhage R R 1993 Query optimization in information retrieval using genetic algorithms *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 603–11

Zhang B-T and Mühlenbein H 1993 Genetic programming of minimal neural nets using Occam's razor *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 342–9

Further reading

This article has provided only a glimpse into the range of applications for evolutionary computing. A series of comprehensive bibliographies has been produced by J T Alander of the Department of Information Technology and Production Economics, University of Vaasa, as listed below.

1. **Art and Music:** *Indexed Bibliography of Genetic Algorithms in Art and Music* Report 94-1-ART (ftp.uwasa.fi/cs/report94-1/gaARTbib.ps.Z)
2. **Chemistry and Physics:** *Indexed Bibliography of Genetic Algorithms in Chemistry and Physics* Report 94-1-CHEMPHYS (ftp.uwasa.fi/cs/report94-1/gaCHEMPHYSbib.ps.Z)
3. **Control:** *Indexed Bibliography of Genetic Algorithms in Control*. Report 94-1-CONTROL (ftp.uwasa.fi/cs/report94-1/gaCONTROLbib.ps.Z)
4. **Computer Aided Design:** *Indexed Bibliography of Genetic Algorithms in Computer Aided Design* Report 94-1-CAD (ftp.uwasa.fi/cs/report94-1/gaCADbib.ps.Z)
5. **Computer Science:** *Indexed Bibliography of Genetic Algorithms in Computer Science* Report 94-1-CS (ftp.uwasa.fi/cs/report94-1/gaCSbib.ps.Z)
6. **Economics:** *Indexed Bibliography of Genetic Algorithms in Economics* Report 94-1-ECO (ftp.uwasa.fi/cs/report94-1/gaECObib.ps.Z)
7. **Electronics and VLSI Design and Testing:** *Indexed Bibliography of Genetic Algorithms in Electronics and VLSI Design and Testing* Report 94-1-VLSI (ftp.uwasa.fi/cs/report94-1/gaVLSIbib.ps.Z)
8. **Engineering:** *Indexed Bibliography of Genetic Algorithms in Engineering* Report 94-1-ENG (ftp.uwasa.fi/cs/report94-1/gaENGbib.ps.Z)
9. **Fuzzy Systems:** *Indexed Bibliography of Genetic Algorithms and Fuzzy Systems* Report 94-1-FUZZY (ftp.uwasa.fi/cs/report94-1/gaFUZZYbib.ps.Z)
10. **Logistics:** *Indexed Bibliography of Genetic Algorithms in Logistics* Report 94-1-LOGISTICS (ftp.uwasa.fi/cs/report94-1/gaLOGISTICSbib.ps.Z)
11. **Manufacturing:** *Indexed Bibliography of Genetic Algorithms in Manufacturing* Report 94-1-MANU (ftp.uwasa.fi/cs/report94-1/gaMANUbib.ps.Z)
12. **Neural Networks:** *Indexed Bibliography of Genetic Algorithms and Neural Networks* Report 94-1-NN (ftp.uwasa.fi/cs/report94-1/gaNNbib.ps.Z)
13. **Optimization:** *Indexed Bibliography of Genetic Algorithms and Optimization* Report 94-1-OPTIMI (ftp.uwasa.fi/cs/report94-1/gaOPTIMIbib.ps.Z)
14. **Operations Research:** *Indexed Bibliography of Genetic Algorithms in Operations Research* Report 94-1-OR (ftp.uwasa.fi/cs/report94-1/gaORbib.ps.Z)
15. **Power Engineering:** *Indexed Bibliography of Genetic Algorithms in Power Engineering* Report 94-1-POWER (ftp.uwasa.fi/cs/report94-1/gaPOWERbib.ps.Z)
16. **Robotics:** *Indexed Bibliography of Genetic Algorithms in Robotics* Report 94-1-ROBO (ftp.uwasa.fi/cs/report94-1/gaROBObib.ps.Z)
17. **Signal and Image Processing:** *Indexed Bibliography of Genetic Algorithms in Signal and Image Processing* Report 94-1-SIGNAL (ftp.uwasa.fi/cs/report94-1/gaSIGNALbib.ps.Z)

A1.3 Advantages (and disadvantages) of evolutionary computation over other approaches

Hans-Paul Schwefel

Abstract

The attractiveness of evolutionary algorithms is obvious from the many successful applications already and the huge number of publications in the field of evolutionary computation. Trying to offer hard facts about comparative advantages in general, however, turns out to be difficult—if not impossible. One reason for this is the so-called no-free-lunch (NFL) theorem.

A1.3.1 No-free-lunch theorem

Since, according to the no-free-lunch (NFL) theorem (Wolpert and Macready 1996), there cannot exist any algorithm for solving *all* (e.g. optimization) problems that is generally (on average) superior to any competitor, the question of whether evolutionary algorithms (EAs) are inferior/superior to any alternative approach is senseless. What could be claimed solely is that EAs behave better than other methods with respect to solving a specific class of problems—with the consequence that they behave worse for other problem classes.

The NFL theorem can be corroborated in the case of EAs versus many classical optimization methods insofar as the latter are more efficient in solving linear, quadratic, strongly convex, unimodal, separable, and many other special problems. On the other hand, EAs do not give up so early when discontinuous, nondifferentiable, multimodal, noisy, and otherwise unconventional response surfaces are involved. Their effectiveness (or robustness) thus extends to a broader field of applications, of course with a corresponding loss in efficiency when applied to the classes of simple problems classical procedures have been specifically devised for.

Looking into the historical record of procedures devised to solve optimization problems, especially around the 1960s (see the book by Schwefel (1995)), when a couple of direct optimum-seeking algorithms were published, for example, in the *Computer Journal*, a certain pattern of development emerges. Author A publishes a procedure and demonstrates its suitability by means of tests using some test functions. Next, author B comes along with a counterexample showing weak performance of A's algorithm in the case of a certain test problem. Of course, he also presents a new or modified technique that outperforms the older one(s) with respect to the additional test problem. This game could in principle be played *ad infinitum*.

A better means of clarifying the scene ought to result from theory. This should clearly define the domain of applicability of each algorithm by presenting convergence proofs and efficiency results. Unfortunately, however, it is possible to prove abilities of algorithms only by simplifying them as well as the situations to which they are confronted. The huge remainder of questions must be answered by means of (always limited) test series, and even that cannot tell much about an actual real-world problem-solving situation with yet unanalyzed features, that is, the normal case in applications.

Again unfortunately, there does not exist an agreed-upon test problem catalogue to evaluate old as well as new algorithms in a concise way. It is doubtful whether such a test bed will ever be agreed upon, but efforts in that direction would be worthwhile.

A1.3.2 Conclusions

Finally, what are the truths and consequences? First, there will always remain a dichotomy between efficiency and general applicability, between reliability and effort of problem-solving, especially optimum-seeking, algorithms. Any specific knowledge about the situation at hand may be used to specify an adequate specific solution algorithm, the optimal situation being that one knows the solution in advance. On the other hand, there cannot exist one method that solves all problems effectively as well as efficiently. These goals are contradictory.

If there is already a traditional method that solves a given problem, EAs should not be used. They cannot do it better or with less computational effort. In particular, they do not offer an escape from the curse of dimensionality—the often quadratic, cubic, or otherwise polynomial increase in instructions used as the number of decision variables is increased, arising, for example, from matrix manipulation.

To develop a new solution method suitable for a problem at hand may be a nice challenge to a theoretician, who will afterwards get some merit for his effort, but from the application point of view the time for developing the new technique has to be added to the computer time invested. In that respect, a nonspecialized, robust procedure (and EAs belong to this class) may be, and often proves to be, worthwhile.

A warning should be given about a common practice—the linearization or other decomplexification of the situation in order to make a traditional method applicable. Even a guaranteed globally optimal solution for the simplified task may be a long way off and thus largely inferior to an approximate solution to the real problem.

The best one can say about EAs, therefore, is that they present a methodological framework that is easy to understand and handle, and is either usable as a black-box method or open to the incorporation of new or old recipes for further sophistication, specialization or hybridization. They are applicable even in dynamic situations where the goal or constraints are moving over time or changing, either exogenously or self-induced, where parameter adjustments and fitness measurements are disturbed, and where the landscape is rough, discontinuous, multimodal, even fractal or cannot otherwise be handled by traditional methods, especially those that need global prediction from local surface analysis.

There exist EA versions for *multiple criteria decision making* (MCDM) and many different parallel computing architectures. Almost forgotten today is their applicability in experimental (non-computing) situations. C4.5, F1.9

Sometimes striking is the fact that even obviously wrong *parameter settings* do not prevent fairly good results: this certainly can be described as robustness. Not yet well understood, but nevertheless very successful are those EAs which self-adapt some of their internal parameters, a feature that can be described as collective learning of the environmental conditions. Nevertheless, even self-adaptation does not circumvent the NFL theorem. E1

In this sense, and only in this sense, EAs always present an intermediate compromise; the enthusiasm of its inventors is not yet taken into account here, nor the insights available from the analysis of the algorithms for natural evolutionary processes which they try to mimic.

References

Schwefel H-P 1995 *Evolution and Optimum Seeking* (New York: Wiley)

Wolpert D H and Macready W G 1996 *No Free Lunch Theorems for Search* Technical Report SFI-TR-95-02-010 Santa Fe Institute

A2.1 Principles of evolutionary processes

David B Fogel

Abstract

The principles of evolution are considered. Evolution is seen to be the inevitable outcome of the interaction of four essential processes: reproduction, competition, mutation, and selection. Consideration is given to the duality of natural organisms in terms of their genotypes and phenotypes, as well as to characterizing evolution in terms of adaptive landscapes.

A2.1.1 Overview

The most widely accepted collection of evolutionary theories is the neo-Darwinian paradigm. These arguments assert that the vast majority of the history of life can be fully accounted for by physical processes operating on and within populations and species (Hoffman 1989, p 39). These processes are reproduction, mutation, competition, and selection. Reproduction is an obvious property of extant species. Further, species have such great reproductive potential that their population size would increase at an exponential rate if all individuals of the species were to reproduce successfully (Malthus 1826, Mayr 1982, p. 479). Reproduction is accomplished through the transfer of an individual's genetic program (either asexually or sexually) to progeny. Mutation, in a positively entropic system, is guaranteed, in that replication errors during information transfer will necessarily occur. Competition is a consequence of expanding populations in a finite resource space. Selection is the inevitable result of competitive replication as species fill the available space. Evolution becomes the inescapable result of interacting basic physical statistical processes (Huxley 1963, Wooldridge 1968, Atmar 1979).

Individuals and species can be viewed as a duality of their genetic program, the *genotype*, and their expressed behavioral traits, the *phenotype*. The genotype provides a mechanism for the storage of experiential evidence, of historically acquired information. Unfortunately, the results of genetic variations are generally unpredictable due to the universal effects of pleiotropy and polygeny (figure A2.1.1) (Mayr 1959, 1963, 1982, 1988, Wright 1931, 1960, Simpson 1949, p 224, Dobzhansky 1970, Stanley 1975, Dawkins 1986). Pleiotropy is the effect that a single gene may simultaneously affect several phenotypic traits. Polygeny is the effect that a single phenotypic characteristic may be determined by the simultaneous interaction of many genes. There are no one-gene, one-trait relationships in naturally evolved systems. The phenotype varies as a complex, nonlinear function of the interaction between underlying genetic structures and current environmental conditions. Very different genetic structures may code for equivalent behaviors, just as diverse computer programs can generate similar functions. A2.2.2

Selection directly acts only on the expressed behaviors of individuals and species (Mayr 1988, pp 477–8). Wright (1932) offered the concept of adaptive topography to describe the fitness of individuals and species (minimally, isolated reproductive populations termed demes). A population of genotypes maps to respective phenotypes (*sensu* Lewontin 1974), which are in turn mapped onto the adaptive topography (figure A2.1.2). Each peak corresponds to an optimized collection of phenotypes, and thus to one of more sets of optimized genotypes. Evolution probabilistically proceeds up the slopes of the topography toward peaks as selection culls inappropriate phenotypic variants.

Others (Atmar 1979, Raven and Johnson 1986, pp 400–1) have suggested that it is more appropriate to view the adaptive landscape from an inverted position. The peaks become troughs, 'minimized prediction

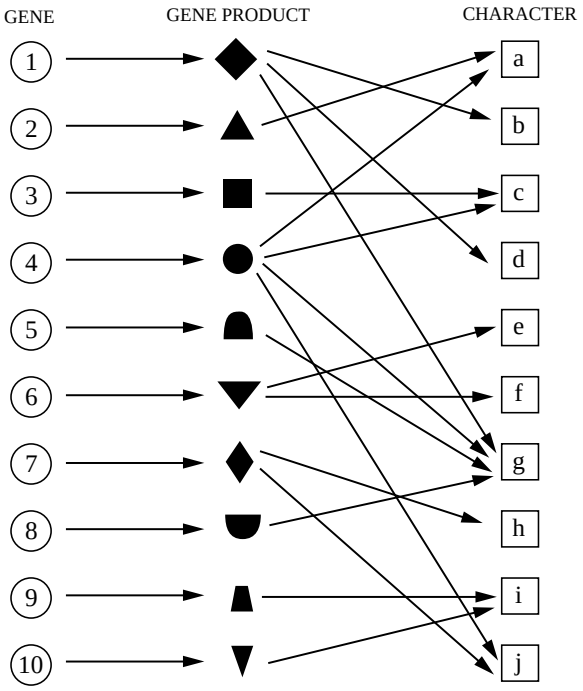


Figure A2.1.1. Pleiotropy is the effect that a single gene may simultaneously affect several phenotypic traits. Polygeny is the effect that a single phenotypic characteristic may be determined by the simultaneous interaction of many genes. These one-to-many and many-to-one mappings are pervasive in natural systems. As a result, even small changes to a single gene may induce a raft of behavioral changes in the individual (after Mayr 1963).

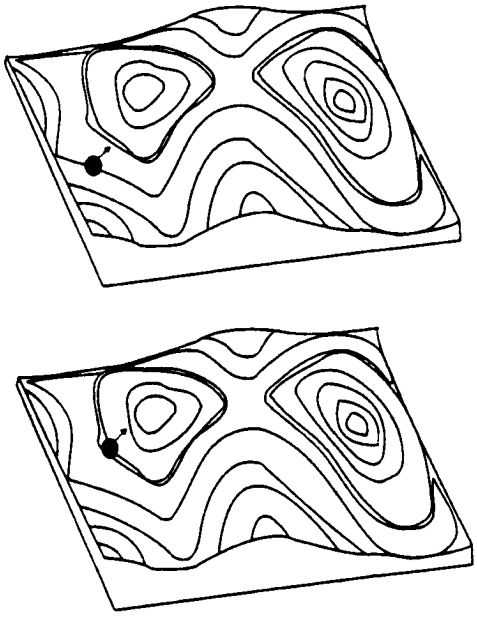


Figure A2.1.2. Wright's adaptive topography, inverted. An adaptive topography, or adaptive landscape, is defined to represent the fitness of all possible phenotypes (generated by the interaction between the genotypes and the environment). Wright (1932) proposed that as selection culls the last appropriate existing behaviors relative to others in the population, the population advances to areas of higher fitness on the landscape. Atmar (1979) and others have suggested viewing the topography from an inverted perspective. Populations advance to areas of lower behavioral error.

error entropy wells' (Atmar 1979). Searching for peaks depicts evolution as a slowly advancing, tedious, uncertain process. Moreover, there appears to be a certain fragility to an evolving phyletic line; an optimized population might be expected to quickly fall of the peak under slight perturbations. The inverted topography leaves an altogether different impression. Populations advance rapidly down the walls of the error troughs until their cohesive set of interrelated behaviors is optimized, at which point stagnation occurs. If the topography is generally static, rapid descents will be followed by long periods of stasis. If, however, the topography is in continual flux, stagnation may never set in.

Viewed in this manner, evolution is an obvious optimizing problem-solving process (not to be confused with a process that leads to perfection). Selection drives phenotypes as close to the optimum as possible, given initial conditions and environment constraints. However the environment is continually changing. Species lag behind, constantly evolving toward a new optimum. No organism should be viewed as being perfectly adapted to its environment. The suboptimality of behavior is to be expected in any dynamic environment that mandates tradeoffs between behavioral requirements. However selection never ceases to operate, regardless of the population's position on the topography.

Mayr (1988, p 532) has summarized some of the more salient characteristics of the neo-Darwinian paradigm. These include:

(i) The individual is the primary target of selection.
(ii) Genetic variation is largely a chance phenomenon. Stochastic processes play a significant role in evolution.
(iii) Genotypic variation is largely a product of recombination and 'only ultimately of mutation'.
(iv) 'Gradual' evolution may incorporate phenotypic discontinuities.
(v) Not all phenotypic changes are necessarily consequences of *ad hoc* natural selection.
(vi) Evolution is a change in adaptation and diversity, not merely a change in gene frequencies.
(vii) Selection is probabilistic, not deterministic.

These characteristics form a framework for evolutionary computation.

References

Atmar W 1979 The inevitability of evolutionary invention, unpublished manuscript

Dawkins R 1986 *The Blind Watchmaker* (Oxford: Clarendon)

Dobzhansky T 1970 *Genetics of the Evolutionary Processes* (New York: Columbia University Press)

Hoffman A 1989 *Arguments on Evolution: a Paleontologist's Perspective* (New York: Oxford University Press)

Huxley J 1963 The evolutionary process *Evolution as a Process* ed J Huxley, A C Hardy and E B Ford (New York: Collier) pp 9–33

Lewontin R C 1974 *The Genetic Basis of Evolutionary Change* (New York: Columbia University Press)

Malthus T R 1826 *An Essay on the Principle of Population, as it Affects the Future Improvement of Society* 6th edn (London: Murray)

Mayr E 1959 Where are we? *Cold Spring Harbor Symp. Quant. Biol.* **24** 409–40

——1963 *Animal Species and Evolution* (Cambridge, MA: Belknap)

——1982 *The Growth of Biological Thought: Diversity, Evolution and Inheritance* (Cambridge, MA: Belknap)

——1988 *Toward a New Philosophy of Biology: Observations of an Evolutionist* (Cambridge, MA: Belknap)

Raven P H and Johnson G B 1986 *Biology* (St Louis, MO: Times Mirror)

Simpson G G 1949 *The Meaning of Evolution: a Study of the History of Life and its Significance for Man* (New Haven, CT: Yale University Press)

Stanley S M 1975 A theory of evolution above the species level *Proc. Natl Acad. Sci. USA* **72** 646–50

Wooldridge D E 1968 *The Mechanical Man: the Physical Basis of Intelligent Life* (New York: McGraw-Hill)

Wright S 1931 Evolution in Mendelian populations *Genetics* **16** 97–159

——1932 The roles of mutation, inbreeding, crossbreeding, and selection in evolution *Proc. 6th Int. Congr. on Genetics (Ithaca, NY)* vol 1, pp 356–66

——1960 The evolution of life, panel discussion *Evolution After Darwin: Issues in Evolution* vol 3, ed S Tax and C Callender (Chicago, IL: University of Chicago Press)

A2.2 Principles of genetics

Raymond C Paton

Abstract

The purpose of this section is to provide the reader with a general overview of the biological background to evolutionary computing. This is not a central issue to understanding how evolutionary algorithms work or how they can be applied. However, many biological terms have been reapplied in evolutionary computing and many researchers seek to introduce new ideas from biological sources. It is hoped to provide valuable background for such readers.

A2.2.1 Introduction

The material covers a number of key areas which are necessary to understanding the nature of the evolutionary process. We begin by looking at some basic ideas of heredity and how variation occurs in interbreeding populations. From here we look at the gene in more detail and then consider how it can undergo change. The next section looks at aspects of population thinking needed to appreciate selection. This is crucial to an appreciation of Darwinian mechanisms of evolution. The article concludes with selected references to further information. In order to keep this contribution within its size limits, the material is primarily about the biology of higher plants and animals.

A2.2.2 Some fundamental concepts in genetics

Many plants and animals are produced through sexual means by which the nucleus of a male sperm cell fuses with a female egg cell (ovum). Sperm and ovum nuclei each contain a single complement of nuclear material arranged as ribbon-like structures called chromosomes. When a sperm fuses with an egg the resulting cell, called a zygote, has a double complement of chromosomes together with the cytoplasm of the ovum. We say that a single complement of chromosomes constitutes a haploid set (abbreviated as n)

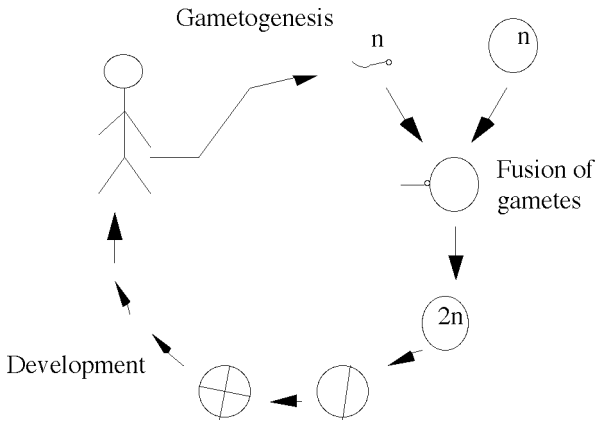


Figure A2.2.1. A common life cycle model.

and a double complement is called the diploid set ($2n$). Gametes (sex cells) are haploid whereas most other cells are diploid. The formation of gametes (gametogenesis) requires the number of chromosomes in the gamete-forming cells to be halved (see figure A2.2.1).

Gametogenesis is achieved through a special type of cell division called meiosis (also called reduction division). The intricate mechanics of meiosis ensures that gametes contain only one copy of each chromosome.

A genotype is the genetic constitution that an organism inherits from its parents. In a diploid organism, half the genotype is inherited from one parent and half from the other. Diploid cells contain two copies of each chromosome. This rule is not universally true when it comes to the distribution of sex chromosomes. Human diploid cells contain 46 chromosomes of which there are 22 pairs and an additional two sex chromosomes. Sex is determined by one pair (called the sex chromosomes); female is X and male is Y. A female human has the sex chromosome genotype of XX and a male is XY. The inheritance of sex is summarized in figure A2.2.2. The members of a pair of nonsex chromosomes are said to be homologous (this is also true for XX genotypes whereas XY are not homologous).

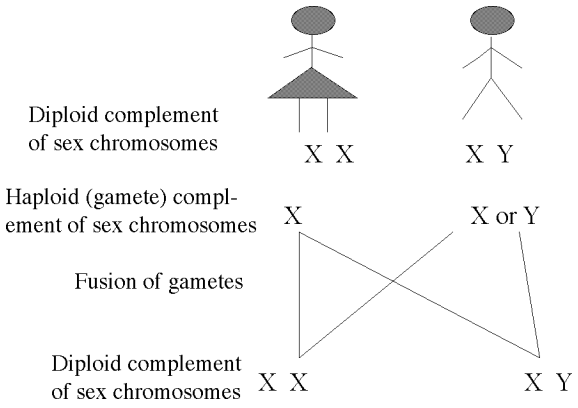


Figure A2.2.2. Inheritance of sex chromosomes.

Although humans have been selectively breeding domestic animals and plants for a long time, the modern study of genetics began in the mid-19th century with the work of Gregor Mendel. Mendel investigated the inheritance of particular traits in peas. For example, he took plants that had wrinkled seeds and plants that had round seeds and bred them with plants of the same phenotype (i.e. observable appearance), so wrinkled were bred with wrinkled and round were bred with round. He continued this over a number of generations until round always produced round offspring and wrinkled, wrinkled. These are called pure breeding plants. He then cross-fertilized the plants by breeding rounds with wrinkles. The subsequent generation (called the F1 hybrids) was all round. Then Mendel crossed the F1 hybrids with each other and found that the next generation, the F2 hybrids, had round and wrinkled plants in the ratio of 3 (round) : 1 (wrinkled).

Mendel did this kind of experiment with a number of pea characteristics such as:

color of cotyledons	yellow or green
color of flowers	red or white
color of seeds	gray/brown or white
length of stem	tall or dwarf.

In each case he found that the the F1 hybrids were always of one form and the two forms reappeared in the F2. Mendel called the form which appeared in the F1 generation dominant and the form which reappeared in the F2 recessive (for the full text of Mendel's experiments see an older genetics book, such as that by Sinnott *et al* (1958)).

A modern interpretation of inheritance depends upon a proper understanding of the nature of a gene and how the gene is expressed in the phenotype. The nature of a gene is quite complex as we shall see later (see also Alberts *et al* 1989, Lewin 1990, Futuyma 1986). For now we shall take it to be the functional unit of inheritance. An allele (allelomorph) is one of several forms of a gene occupying a given locus (location) on a chromosome. Originally related to pairs of contrasting characteristics (see examples

above), the idea of observable unit characters was introduced to genetics around the turn of this century by such workers as Bateson, de Vries, and Correns (see Darden 1991). The concept of a gene has tended to replace allele in general usage although the two terms are not the same.

How can the results of Mendel's experiments be interpreted? We know that each parent plant provides half the chromosome complement found in its offspring and that chromosomes in the diploid cells are in pairs of homologues. In the pea experiments pure breeding parents had homologous chromosomes which were identical for a particular gene; we say they are homozygous for a particular gene. The pure breeding plants were produced through self-fertilization and by selecting those offspring of the desired phenotype. As round was dominant to wrinkled we say that the round form of the gene is R ('big r') and the wrinkled r ('little r'). Figure A2.2.3 summarizes the cross of a pure breeding round (RR) with a pure breeding wrinkled (rr).

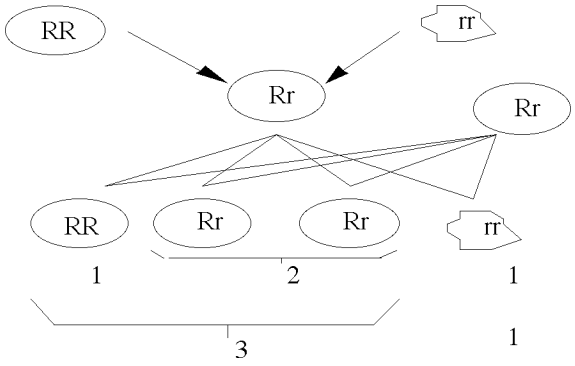


Figure A2.2.3. A simple Mendelian experiment.

We see the appearance of the heterozygote (in this case Rr) in the F1 generation. This is phenotypically the same as the dominant phenotype but genotypically contains both a dominant and a recessive form of the particular gene under study. Thus when the heterozygotes are randomly crossed with each other the phenotype ratio is three dominant : one recessive. This is called the monohybrid ratio (i.e. for one allele). We see in Mendel's experiments the independent segregation of alleles during breeding and their subsequent independent assortment in offspring.

In the case of two genes we find more phenotypes and genotypes appearing. Consider what happens when pure breeding homozygotes for round yellow seeds (RRYY) are bred with pure breeding homozygotes for wrinkled green seeds (rryy). On being crossed we end up with heterozygotes with a genotype of RrYy and phenotype of round yellow seeds. We have seen that the genes segregate independently during meiosis so we have the combinations shown in figure A2.2.4.

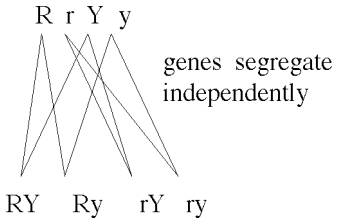


Figure A2.2.4. Genes segregating independently.

Thus the gametes of the heterozygote can be of four kinds though we assume that each form can occur with equal frequency. We may examine the possible combinations of gametes for the next generation by producing a contingency table for possible gamete combinations. These are shown in figure A2.2.5.

We summarize this set of genotype combinations in the phenotype table (figure A2.2.5(*b*)). The resulting ratio of phenotypes is called the dihybrid ratio (9:3:3:1). We shall consider one final example in this very brief summary. When pure breeding red-flowered snapdragons were crossed with pure breeding white-flowered plants the F1 plants were all pink. When these were selfed the population of offspring was in the ratio of one red : two pink : one white. This is a case of incomplete dominance in the heterozygote.

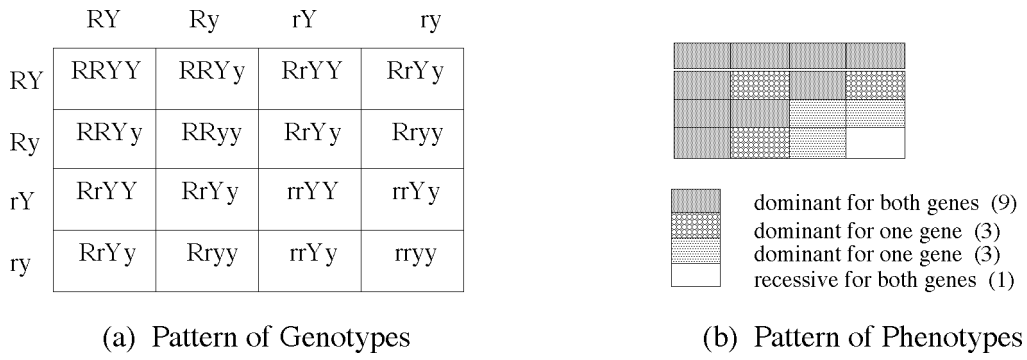


Figure A2.2.5. Genotype and phenotype patterns in F2.

It has been found that the Mendelian ratios do not always apply in breeding experiments. In some cases this is because certain genes interact with each other. Epistasis occurs when the expression of one gene masks the phenotypic effects of another. For example, certain genotypes (cyanogenics) of clover can resist grazing because they produce low doses of cyanide which makes them unpalatable. Two genes are involved in cyanide production, one which produces an enzyme which converts a precursor molecule into a glycoside and another gene which produces an enzyme which converts the glycoside into hydrogen cyanide (figure A2.2.6(*a*)). If two pure breeding acyanogenic strains are crossed the heterozygote is cyanogenic (figure A2.2.6(*b*)).

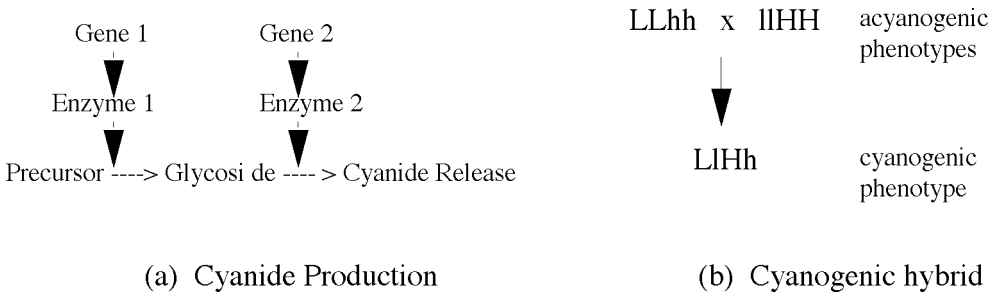


Figure A2.2.6. Cyanogenic clover: cyanide production and cyanogenic hybrid.

When the cyanogenic strain is selfed the genotypes are as summarized in figure A2.2.7(*a*). There are only two phenotypes produced, cyanogenic and acyanogenic, as summarized in figure A2.2.7(*b*).

So far we have followed Mendel's laws regarding the independent segregation of genes. This independent segregation does not occur when genes are located on the same chromosome. During meiosis homologous chromosomes (i.e. matched pairs one from each parental gamete) move together and are seen to be joined at the centromere (the clear oval region in figure A2.2.8).

In this simplified diagram we show a set of genes (rectangles) in which those on the top are of the opposite form to those on the bottom. As the chromosomes are juxtaposed they each are doubled up so

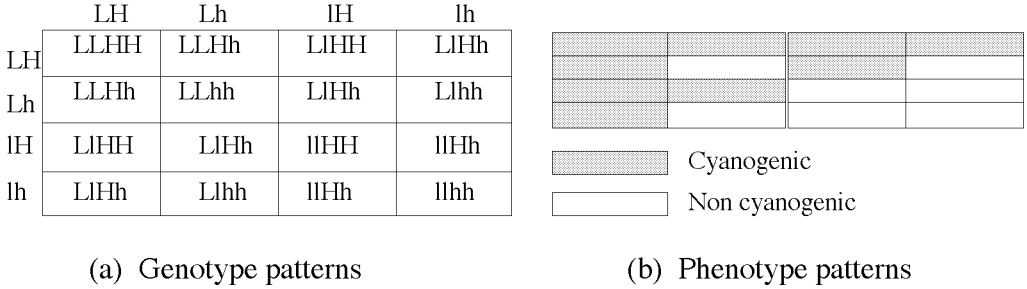


Figure A2.2.7. Epistasis in clover: genotype and phenotype patterns.

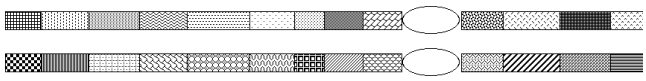


Figure A2.2.8. Pairing of homologous chromosomes.

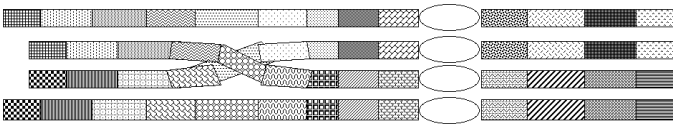


Figure A2.2.9. Crossing over in a tetrad.

that four strands (usually called chromatids) are aligned. The close proximity of the inner two chromatids and the presence of enzymes in the cellular environment can result in breakages and recombinations of these strands as summarized in figure A2.2.9.

The result is that of the four juxtaposed strands two are the same as the parental chromosomes and two, called the recombinants, are different. This crossover process mixes up the genes with respect to original parental chromosomes. The chromosomes which make up a haploid gamete will be a random mixture of parental and recombinant forms. This increases the variability between parents and offspring and reduces the chance of harmful recessives becoming homozygous.

A2.2.3 The gene in more detail

Genes are located on chromosomes. Chromosomes segregate independently during meiosis whereas genes can be linked on the same chromosome. The conceptual reasons why there has been confusion are the differences in understanding about gene and chromosome such as which is the unit of heredity (see Darden 1991). The discovery of the physicochemical nature of hereditary material culminated in the Watson–Crick model in 1953 (see figure A2.2.10). The coding parts of the deoxyribonucleic acid (DNA) are called bases; there are four types (adenine, thymine, cytosine, and guanine). They are strung along a sugar-and-phosphate string, which is arranged as a helix. Two intertwined strings then form the double helix. The functional unit of this code is a triplet of bases which can code for a single amino acid. The genes are located along the DNA strand.

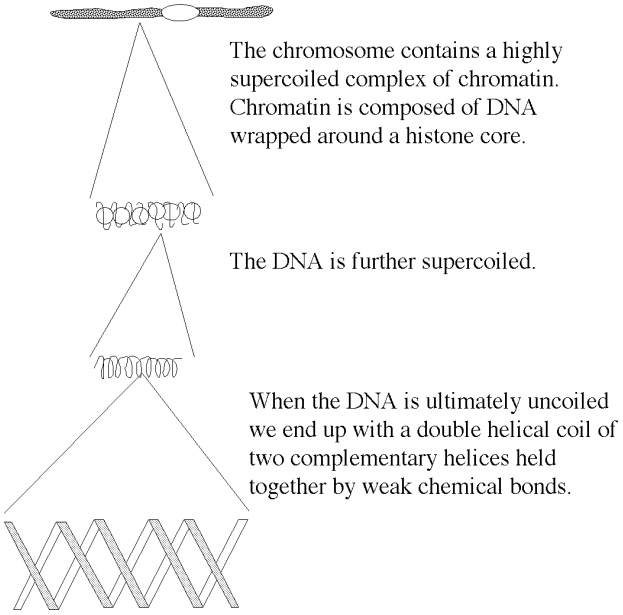


Figure A2.2.10. Idealization of the organization of chromosomes in a eukaryotic cell. (A eukaryotic cell has an organized nucleus and cytoplasmic organelles.)

Transcription is the synthesis of ribonucleic acid (RNA) using the DNA template. It is a preliminary step in the ultimate synthesis of protein. A gene can be transcribed through the action of enzymes and a chain of transcript is formed as a polymer called messenger RNA (mRNA). This mRNA can then be translated into protein. The translation process converts the mRNA code into a protein sequence via another form of RNA called transfer RNA (tRNA). In this way, genes are transcribed so that mRNA may be produced, from which protein molecules (typically the 'workhorses' and structural molecules of a cell) can be formed. This flow of information is generally unidirectional. (For more details on this topic the reader should consult a molecular biology text and look at the central dogma of molecular biology, see e.g. Lewin 1990, Alberts *et al* 1989.)

Figure A2.2.11 provides a simplified view of the anatomy of a structural gene, that is, one which codes for a protein or RNA.

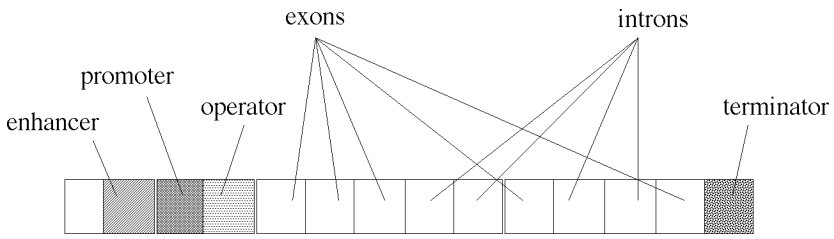


Figure A2.2.11. A simplified diagram of a structural gene.

That part of the gene which ultimately codes for protein or RNA is preceded upstream by three stretches of code. The enhancer facilitates the operation of the promoter region, which is where RNA polymerase is bound to the gene in order to initiate transcription. The operator is the site where transcription can be halted by the presence of a repressor protein. Exons are expressed in the final gene product (e.g. the protein molecule) whereas introns are transcribed but are removed from the transcript leaving the fragments of exon material to be spliced. One stretch of DNA may consist of several overlapping genes. For example, the introns in one gene may be the exons in another (Lewin 1990). The terminator is the postexon region of the gene which causes transcription to be terminated. Thus a biological gene contains not only code to be read but also coded instructions on how it should be read and what should be read. Genes are highly organized. An operon system is located on one chromosome and consists of a regulator gene and a number of contiguous structural genes which share the same promoter and terminator and code for enzymes which are involved in specific metabolic pathways (the classical example is the Lac operon, see figure A2.2.12).

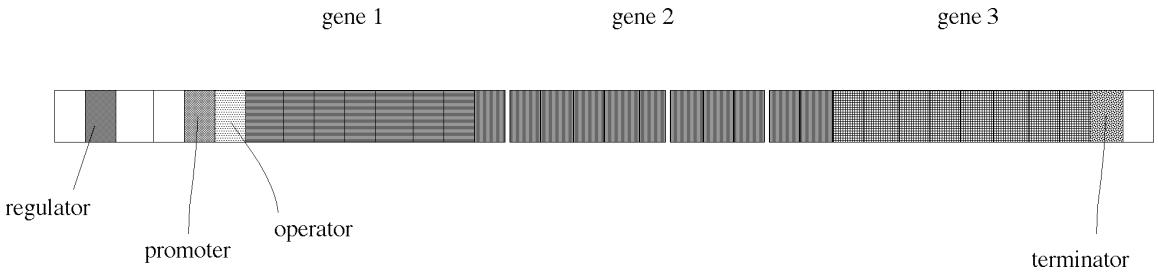


Figure A2.2.12. A visualization of an operon.

Operons can be grouped together into higher-order (hierarchical) regulatory genetic systems (Neidhart *et al* 1990). For example, a number of operons from different chromosomes may be regulated by a single gene known as a regulon. These higher-order systems provide a great challenge for change in a genome. Modification of the higher-order gene can have profound effects on the expression of structural genes that are under its influence.

A2.2.4 Options for change

We have already seen how sexual reproduction can mix up the genes which are incorporated in a gamete through the random reassortment of paternal and maternal chromosomes and through crossing over and recombination. Effectively though, the gamete acquires a subset of the same genes as the diploid gamete-producing cells; they are just mixed up. Clearly, any zygote that is produced will have a mixture of genes and (possibly) some chromosomes which have both paternal and maternal genes.

There are other mechanisms of change which alter the genes themselves or change the number of genes present in a genome. We shall describe a mutation as any change in the sequence of genomic DNA. Gene mutations are of two types: point mutation, in which a single base is changed, and frameshift mutation, in which one or more bases (but not a multiple of three) are inserted or deleted. This changes the frame in which triplets are transcribed into RNA and ultimately translated into protein. In addition some genes are able to become transposed elsewhere in a genome. They 'jump' about and are called transposons. Chromosome changes can be caused by deletion (loss of a section), duplication (the section is repeated), inversion (the section is in the reverse order), and translocation (the section has been relocated elsewhere). There are also changes at the genome level. Ploidy is the term used to describe multiples of a chromosome complement such as haploid (n), diploid ($2n$), and tetraploid ($4n$). A good example of the influence of ploidy on evolution is among such crops as wheat and cotton. Somy describes changes to the frequency of particular chromosomes: for example, trisomy is three copies of a chromosome.

A2.2.5 Population thinking

So far we have focused on how genes are inherited and how they or their combinations can change. In order to understand *evolutionary processes* we must shift our attention to looking at populations (we shall not emphasize too much whether of genes, chromosomes, genomes, or organisms). Population thinking is central to our understanding of models of evolution. A2.1

The Hardy–Weinberg theorem applies to frequencies of genes and genotypes in a population of individuals, and states that the relative frequency of each gene remains in equilibrium from one generation to the next. For a single allele, if the frequency of one form is p then that of the other (say q) is $1 - p$. The three genotypes that exist with this allele have the population proportions of

$$p^2 + 2pq + q^2 = 1.$$

This equation does not apply when a mixture of four factors changes the relative frequencies of genes in a population: mutation, selection, gene flow, and random genetic drift (drift). Drift can be described as the effect of the sampling of a population on its parents. Each generation can be thought of as a sample of its parents' population. In that the current population is a sample of its parents, we acknowledge that a statistical sampling error should be associated with gene frequencies. The effect will be small in large populations because the relative proportion of random changes will be a very small component of the large numbers. However, drift in a small population will have a marked effect.

One factor which can counteract the effect of drift is differential migration of individuals between populations which leads to gene flow. Several models of gene flow exist. For example, migration which occurs at random among a group of small populations is called the *island model* whereas in the *stepping stone* model each population receives migrants only from neighboring populations. Mutation, selection, and gene flow are deterministic factors so that if fitness, mutation rate, and rate of gene flow are the same for a number of populations that begin with same gene frequencies, they will attain the same equilibrium composition. Drift is a stochastic process because the sampling effect on the parent population is random. C6.3

Sewall Wright introduced the idea of an *adaptive landscape* to explain how a population's allele frequencies might evolve over time. The peaks on the landscape represent genetic compositions of a population for which the mean fitness is high and troughs are possible compositions where the mean fitness is low. As gene frequencies change and mean fitness increases the population moves uphill. Indeed, selection will operate to increase mean fitness so, on a multipeaked landscape, selection may operate to move populations to local maxima. On a fixed landscape drift and selection can act together so that populations may move uphill (through selection) or downhill (through drift). This means that the global maximum for the landscape could be reached. These ideas are formally encapsulated in Wright's (1968–1978) *shifting balance theory* of evolution. Further information on the relation of population genetics to B2.7

evolutionary theory can be studied further in the books by Wright (1968–1978), Crow and Kimura (1970) and Maynard Smith (1989).

The change of gene frequencies coupled with changes in the genes themselves can lead to the emergence of new species although the process is far from simple and not fully understood (Futuyma, 1986, Maynard Smith 1993). The nature of the species concept or (for some) concepts which is central to Darwinism is complicated and will not be discussed here (see e.g. Futuyma 1986). Several mechanisms apply to promote speciation (Maynard Smith 1993): geographical or spatial isolation, barriers preventing formation of hybrids, nonviable hybrids, hybrid infertility, and hybrid breakdown—in which post-F1 generations are weak or infertile.

Selectionist theories emphasize invariant properties of the system: the system is an internal generator of variations (Changeux and Dehaene 1989) and diversity among units of the population exists prior to any testing (Manderick 1994). We have seen how section operates to optimize fitness. Darden and Cain (1987) summarize a number of common elements in selectionist theories as follows:

- a set of a given entity type (i.e. the units of the population)
- a particular property (P) according to which members of this set vary
- an environment in which the entity type is found
- a factor in the environment to which members react differentially due to their possession or nonpossession of the property (P)
- differential benefits (both shorter and longer term) according to the possession or nonpossession of the property (P).

This scheme summarizes the selectionist approach. In addition, Maynard Smith (1989) discusses a number of selection systems (particular relevant to animals) including sexual, habitat, family, kin, group, and synergistic (cooperation). A very helpful overview of this area of ecology, behavior, and evolution is that by Sigmund (1993). Three selectionist systems in the biosciences are the neo-Darwinian theory of evolution in a population, clonal selection theory applied to the immune system, and the theory of neuronal group selection (for an excellent summary with plenty of references see that by Manderick (1994)).

There are many important aspects of evolutionary biology which have had to be omitted because of lack of space. The relevance of neutral molecular evolution theory (Kimura 1983) and nonselectionist approaches (see e.g. Goodwin and Saunders 1989, Lima de Faria 1988, Kauffman 1993) has not been discussed. In addition some important ideas have not been considered, such as evolutionary game theory (Maynard Smith 1989, Sigmund 1993), the role of sex (see e.g. Hamilton *et al* 1990), the evolution of cooperation (Axelrod 1984), the red queen (Van Valen 1973, Maynard Smith 1989), structured genomes, for example, incorporation of regulatory hierarchies (Kauffman 1993, Beaumont 1993, Clarke *et al* 1993), experiments with endosymbiotic systems (Margulis and Foster 1991, Hilario and Gogarten 1993), coevolving parasite populations (see e.g. Collins 1994; for a biological critique and further applications see Sumida and Hamilton 1994), inheritance of acquired characteristics (Landman 1991), and genomic imprinting and other epigenetic inheritance systems (for a review see Paton 1994). There are also considerable philosophical issues which must be addressed in this area which impinge on how biological sources are applied to evolutionary computing (see Sober 1984). Not least among these is the nature of adaptation.

References

Alberts B, Bray D, Lewis J, Raff M, Roberts K and Watson J D 1989 *Molecular Biology of the Cell* (New York: Garland)

Axelrod R 1984 *The Evolution of Co-operation* (Harmondsworth: Penguin)

Beaumont M A 1993 Evolution of optimal behaviour in networks of Boolean automata *J. Theor. Biol.* **165** 455–76

Changeux J-P and Dehaene S 1989 Neuronal models of cognitive functions *Cognition* **33** 63-109

Clarke B, Mittenthal J E and Senn M 1993 A model for the evolution of networks of genes *J. Theor. Biol.* **165** 269–89

Collins R 1994 Artificial evolution and the paradox of sex *Computing with Biological Metaphors* ed R C Paton (London: Chapman and Hall)

Crow J F and Kimura M 1970 *An Introduction to Population Genetics Theory* (New York: Harper and Row)

Darden L 1991 *Theory Change in Science* (New York: Oxford University Press)

Darden L and Cain J A 1987 Selection type theories *Phil. Sci.* **56** 106–29

Futuyma D J 1986 *Evolutionary Biology* (MA: Sinauer)

Goodwin B C and Saunders P T (eds) 1989 *Theoretical Biology: Epigenetic and Evolutionary Order from Complex Systems* (Edinburgh: Edinburgh University Press)

Hamilton W D, Axelrod A and Tanese R 1990 Sexual reproduction as an adaptation to resist parasites *Proc. Natl Acad. Sci. USA* **87** 3566–73

Hilario E and Gogarten J P 1993 Horizontal transfer of ATPase genes—the tree of life becomes a net of life *BioSystems* **31** 111–9

Kauffman S A 1993 *The Origins of Order* (New York: Oxford University Press)

Kimura, M 1983 *The Neutral Theory of Molecular Evolution* (Cambridge: Cambridge University Press)

Landman O E 1991 The inheritance of acquired characteristics *Ann. Rev. Genet.* **25** 1–20

Lewin B 1990 *Genes IV* (Oxford: Oxford University Press)

Lima de Faria A 1988 *Evolution without Selection* (Amsterdam: Elsevier)

Margulis L and Foster R (eds) 1991 *Symbiosis as a Source of Evolutionary Innovation: Speciation and Morphogenesis* (Cambridge, MA: MIT Press)

Manderick B 1994 The importance of selectionist systems for cognition *Computing with Biological Metaphors* ed R C Paton (London: Chapman and Hall)

Maynard Smith J 1989 *Evolutionary Genetics* (Oxford: Oxford University Press)

——1993 *The Theory of Evolution* Canto edn (Cambridge: Cambridge University Press)

Neidhart F C, Ingraham J L and Schaechter M 1990 *Physiology of the Bacterial Cell* (Sunderland, MA: Sinauer)

Paton R C 1994 Enhancing evolutionary computation using analogues of biological mechanisms *Evolutionary Computing (Lecture Notes in Computer Science 865)* ed T C Fogarty (Berlin: Springer) pp 51–64

Sigmund K 1993 *Games of Life* (Oxford: Oxford University Press)

Sinnott E W, Dunn L C and Dobzhansky T 1958 *Principles of Genetics* (New York: McGraw-Hill)

Sober E 1984 *The Nature of Selection: Evolutionary Theory in Philosophical Focus* (Chicago, IL: University of Chicago Press)

Sumida B and Hamilton W D 1994 Both Wrightian and 'parasite' peak shifts enhance genetic algorithm performance in the travelling salesman problem *Computing with Biological Metaphors* ed R C Paton (London: Chapman and Hall)

Van Valen L 1973 A new evolutionary law *Evolutionary Theory* **1** 1–30

Wright S 1968–1978 *Evolution and the Genetics of Populations* vols 1–4 (Chicago, IL: Chicago University Press)

A2.3 A history of evolutionary computation

Kenneth De Jong, David B Fogel and Hans-Paul Schwefel

Abstract

This section presents a brief but comprehensive summary of the history of evolutionary computation, starting with the ground-breaking work of the 1950s, tracing the rise and development of the three primary forms of evolutionary algorithms (evolution strategies, evolutionary programming and genetic algorithms), and concluding with the present time, in which a more unified view of the field is emerging.

A2.3.1 Introduction

No one will ever produce a completely accurate account of a set of past events since, as someone once pointed out, writing history is as difficult as forecasting. Thus we dare to begin our historical summary of evolutionary computation rather arbitrarily at a stage as recent as the mid-1950s.

At that time there was already evidence of the use of digital computer models to better understand the natural process of evolution. One of the first descriptions of the use of an evolutionary process for computer problem solving appeared in the articles by Friedberg (1958) and Friedberg *et al* (1959). This represented some of the early work in machine learning and described the use of an evolutionary algorithm for *automatic programming*, i.e. the task of finding a program that calculates a given input–output function. Other founders in the field remember a paper of Fraser (1957) that influenced their early work, and there may be many more such forerunners depending on whom one asks.

In the same time frame Bremermann presented some of the first attempts to apply simulated evolution to numerical optimization problems involving both linear and convex optimization as well as the solution of nonlinear simultaneous equations (Bremermann 1962). Bremermann also developed some of the early evolutionary algorithm (EA) theory, showing that the optimal mutation probability for linearly separable problems should have the value of $1/\ell$ in the case of ℓ bits encoding an individual (Bremermann *et al* 1965).

Also during this period Box developed his *evolutionary operation* (EVOP) ideas which involved an evolutionary technique for the design and analysis of (industrial) experiments (Box 1957, Box and Draper 1969). Box's ideas were never realized as a computer algorithm, although Spendley *et al* (1962) used them as the basis for their so-called *simplex design* method. It is interesting to note that the REVOP proposal (Satterthwaite 1959a, b) introducing randomness into the EVOP operations was rejected at that time.

As is the case with many ground-breaking efforts, these early studies were met with considerable skepticism. However, by the mid-1960s the bases for what we today identify as the three main forms of EA were clearly established. The roots of *evolutionary programming* (EP) were laid by Lawrence Fogel B1.4
in San Diego, California (Fogel *et al* 1966) and those of *genetic algorithms* (GAs) were developed at B1.2
the University of Michigan in Ann Arbor by Holland (1967). On the other side of the Atlantic Ocean, *evolution strategies* (ESs) were a joint development of a group of three students, Bienert, Rechenberg, and B1.3
Schwefel, in Berlin (Rechenberg 1965).

Over the next 25 years each of these branches developed quite independently of each other, resulting in unique parallel histories which are described in more detail in the following sections. However, in 1990 there was an organized effort to provide a forum for interaction among the various EA research

communities. This took the form of an international workshop entitled *Parallel Problem Solving from Nature* at Dortmund (Schwefel and Männer 1991).

Since that event the interaction and cooperation among EA researchers from around the world has continued to grow. In the subsequent years special efforts were made by the organizers of *ICGA'91* (Belew and Booker 1991), *EP'92* (Fogel and Atmar 1992), and *PPSN'92* (Männer and Manderick 1992) to provide additional opportunities for interaction.

This increased interaction led to a consensus for the name of this new field, *evolutionary computation* (EC), and the establishment in 1993 of a journal by the same name published by MIT Press. The increasing interest in EC was further indicated by the *IEEE World Congress on Computational Intelligence (WCCI)* at Orlando, Florida, in June 1994 (Michalewicz *et al* 1994), in which one of the three simultaneous conferences was dedicated to EC along with conferences on neural networks and fuzzy systems. The dramatic growth of interest provided additional evidence for the need of an organized EC handbook (which you are now reading) to provide a more cohesive view of the field.

That brings us to the present in which the continued growth of the field is reflected by the many EC events and related activities each year, and its growing maturity reflected by the increasing number of books and articles about EC.

In order to keep this overview brief, we have deliberately suppressed many of the details of the historical developments within each of the three main EC streams. For the interested reader these details are presented in the following sections.

A2.3.2 Evolutionary programming

Evolutionary programming (EP) was devised by Lawrence J Fogel in 1960 while serving at the National Science Foundation (NSF). Fogel was on leave from Convair, tasked as special assistant to the associate director (research), Dr Richard Bolt, to study and write a report on investing in basic research. Artificial intelligence at the time was mainly concentrated around heuristics and the simulation of primitive neural networks. It was clear to Fogel that both these approaches were limited because they model humans rather than the essential process that produces creatures of increasing intellect: evolution. Fogel considered intelligence to be based on adapting behavior to meet goals in a range of environments. In turn, prediction was viewed as the key ingredient to intelligent behavior and this suggested a series of experiments on the use of simulated evolution of *finite-state machines* to forecast nonstationary time series with respect to arbitrary criteria. These and other experiments were documented in a series of publications (Fogel 1962, 1964, Fogel *et al* 1965, 1966, and many others). C1.5

Intelligent behavior was viewed as requiring the composite ability to (i) predict one's environment, coupled with (ii) a translation of the predictions into a suitable response in light of the given goal. For the sake of generality, the environment was described as a sequence of symbols taken from a finite alphabet. The evolutionary problem was defined as evolving an algorithm (essentially a program) that would operate on the sequence of symbols thus far observed in such a manner so as to produce an output symbol that is likely to maximize the algorithm's performance in light of both the next symbol to appear in the environment and a well-defined payoff function. Finite-state machines provided a useful representation for the required behavior.

The proposal was as follows. A population of finite-state machines is exposed to the environment, that is, the sequence of symbols that have been observed up to the current time. For each parent machine, as each input symbol is offered to the machine, each output symbol is compared with the next input symbol. The worth of this prediction is then measured with respect to the payoff function (e.g. all–none, absolute error, squared error, or any other expression of the meaning of the symbols). After the last prediction is made, a function of the payoff for each symbol (e.g. average payoff per symbol) indicates the fitness of the machine.

Offspring machines are created by randomly mutating each parent machine. Each parent produces offspring (this was originally implemented as only a single offspring simply for convenience). There are five possible modes of random mutation that naturally result from the description of the machine: change an output symbol, change a state transition, add a state, delete a state, or change the initial state. The deletion of a state and change of the initial state are only allowed when the parent machine has more than one state. Mutations are chosen with respect to a probability distribution, which is typically uniform. The number of mutations per offspring is also chosen with respect to a probability distribution or may be fixed *a priori*. These offspring are then evaluated over the existing environment in the same manner as

their parents. Other mutations, such as majority logic mating operating on three or more machines, were proposed by Fogel *et al* (1966) but not implemented.

The machines that provide the greatest payoff are retained to become parents of the next generation. (Typically, half the total machines were saved so that the parent population remained at a constant size.) This process is iterated until an actual prediction of the next symbol (as yet unexperienced) in the environment is required. The best machine generates this prediction, the new symbol is added to the experienced environment, and the process is repeated. Fogel (1964) (and Fogel *et al* (1966)) used 'nonregressive' evolution. To be retained, a machine had to rank in the best half of the population. Saving lesser-adapted machines was discussed as a possibility (Fogel *et al* 1966, p 21) but not incorporated.

This general procedure was successfully applied to problems in prediction, identification, and automatic control (Fogel *et al* 1964, 1966, Fogel 1968) and was extended to simulate coevolving populations by Fogel and Burgin (1969). Additional experiments evolving finite-state machines for sequence prediction, pattern recognition, and gaming can be found in the work of Lutter and Huntsinger (1969), Burgin (1969), Atmar (1976), Dearholt (1976), and Takeuchi (1980).

In the mid-1980s the general EP procedure was extended to alternative representations including ordered lists for the traveling salesman problem (Fogel and Fogel 1986), and real-valued vectors for continuous function optimization (Fogel and Fogel 1986). This led to other applications in route planning (Fogel 1988, Fogel and Fogel 1988), optimal subset selection (Fogel 1989), and training neural networks (Fogel *et al* 1990), as well as comparisons to other methods of *simulated evolution* (Fogel and Atmar 1990). F1.10
Methods for extending evolutionary search to a two-step process including evolution of the mutation variance were offered by Fogel *et al* (1991, 1992). Just as the proper choice of step sizes is a crucial part of every numerical process, including optimization, the internal adaptation of the mutation variance(s) is of utmost importance for the algorithm's efficiency. This process is called *self-adaptation* or autoadaptation C7.1
in the case of no explicit control mechanism, e.g. if the variances are part of the individuals' characteristics and underlie probabilistic variation in a similar way as do the ordinary decision variables.

In the early 1990s efforts were made to organize annual conferences on EP, these leading to the first conference in 1992 (Fogel and Atmar 1992). This conference offered a variety of optimization applications of EP in robotics (McDonnell *et al* 1992, Andersen *et al* 1992), path planning (Larsen and Herman 1992, Page *et al* 1992), *neural network design and training* (Sebald and Fogel 1992, Porto 1992, D1
McDonnell 1992), *automatic control* (Sebald *et al* 1992), and other fields. F1.3

First contacts were made between the EP and ES communities just before this conference, and the similar but independent paths that these two approaches had taken to simulating the process of evolution were clearly apparent. Members of the ES community have participated in all successive EP conferences (Bäck *et al* 1993, Sprave 1994, Bäck and Schütz 1995, Fogel *et al* 1996). There is less similarity between EP and GAs, as the latter emphasize simulating specific mechanisms that apply to natural genetic systems whereas EP emphasizes the behavioral, rather than genetic, relationships between parents and their offspring. Members of the GA and GP communities have, however, also been invited to participate in the annual conferences, making for truly interdisciplinary interaction (see e.g. Altenberg 1994, Land and Belew 1995, Koza and Andre 1996).

Since the early 1990s, efforts in EP have diversified in many directions. Applications in training neural networks have received considerable attention (see e.g. English 1994, Angeline *et al* 1994, McDonnell and Waagen 1994, Porto *et al* 1995), while relatively less attention has been devoted to evolving *fuzzy* D2
systems (Haffner and Sebald 1993, Kim and Jeon 1996). Image processing applications can be found in the articles by Bhattacharjya and Roysam (1994), Brotherton *et al* (1994), Rizki *et al* (1995), and others. Recent efforts to use EP in medicine have been offered by Fogel *et al* (1995) and Gehlhaar *et al* (1995). Efforts studying and comparing methods of self-adaptation can be found in the articles by Saravanan *et al* (1995), Angeline *et al* (1996), and others. Mathematical analyses of EP have been summarized by Fogel (1995).

To offer a summary, the initial efforts of L J Fogel indicate some of the early attempts to (i) use simulated evolution to perform prediction, (ii) include variable-length encodings, (iii) use representations that take the form of a sequence of instructions, (iv) incorporate a population of candidate solutions, and (v) coevolve evolutionary programs. Moreover, Fogel (1963, 1964) and Fogel *et al* (1966) offered the early recognition that natural evolution and the human endeavor of the scientific method are essentially similar processes, a notion recently echoed by Gell-Mann (1994). The initial prescriptions for operating on finite-state machines have been extended to arbitrary representations, mutation operators, and selection methods, and techniques for self-adapting the evolutionary search have been proposed and implemented.

The population size need not be kept constant and there can be a variable number of offspring per parent, much like the *$(\mu + \lambda)$ methods* offered in ESs. In contrast to these methods, selection is often made probabilistic in EP, giving lesser-scoring solutions some probability of surviving as parents into the next generation. In contrast to GAs, no effort is made in EP to support (some say maximize) *schema processing*, nor is the use of random variation constrained to emphasize specific mechanisms of genetic transfer, perhaps providing greater versatility to tackle specific problem domains that are unsuitable for genetic operators such as crossover. C2.4.4 B2.5

A2.3.3 Genetic algorithms

The first glimpses of the ideas underlying genetic algorithms (GAs) are found in Holland's papers in the early 1960s (see e.g. Holland 1962). In them Holland set out a broad and ambitious agenda for understanding the underlying principles of adaptive systems—systems that are capable of self-modification in response to their interactions with the environments in which they must function. Such a theory of adaptive systems should facilitate both the understanding of complex forms of adaptation as they appear in natural systems and our ability to design robust adaptive artifacts.

In Holland's view the key feature of robust natural adaptive systems was the successful use of competition and innovation to provide the ability to dynamically respond to unanticipated events and changing environments. Simple models of biological evolution were seen to capture these ideas nicely via notions of survival of the fittest and the continuous production of new offspring.

This theme of using evolutionary models both to understand natural adaptive systems and to design robust adaptive artifacts gave Holland's work a somewhat different focus than those of other contemporary groups that were exploring the use of evolutionary models in the design of efficient experimental optimization techniques (Rechenberg 1965) or for the evolution of intelligent agents (Fogel *et al* 1966), as reported in the previous section.

By the mid-1960s Holland's ideas began to take on various computational forms as reflected by the PhD students working with Holland. From the outset these systems had a distinct 'genetic' flavor to them in the sense that the objects to be evolved over time were represented internally as 'genomes' and the mechanisms of reproduction and inheritance were simple abstractions of familiar population genetics operators such as mutation, crossover, and inversion.

Bagley's thesis (Bagley 1967) involved tuning sets of weights used in the evaluation functions of game-playing programs, and represents some of the earliest experimental work in the use of diploid representations, the role of inversion, and selection mechanisms. By contrast Rosenberg's thesis (Rosenberg 1967) has a very distinct flavor of simulating the evolution of a simple biochemical system in which single-celled organisms capable of producing enzymes were represented in diploid fashion and were evolved over time to produce appropriate chemical concentrations. Of interest here is some of the earliest experimentation with adaptive crossover operators.

Cavicchio's thesis (Cavicchio 1970) focused on viewing these ideas as a form of adaptive search, and tested them experimentally on difficult search problems involving subroutine selection and pattern recognition. In his work we see some of the early studies on *elitist* forms of selection and ideas for adapting the rates of crossover and mutation. Hollstien's thesis (Hollstien 1971) took the first detailed look at alternate selection and mating schemes. Using a test suite of two-dimensional *fitness landscapes*, he experimented with a variety of breeding strategies drawn from techniques used by animal breeders. Also of interest here is Hollstien's use of binary string encodings of the genome and early observations about the virtues of Gray codings. C2.7.4 B2.7.4

In parallel with these experimental studies, Holland continued to work on a general theory of adaptive systems (Holland 1967). During this period he developed his now famous *schema analysis* of adaptive systems, relating it to the optimal allocation of trials using k-armed bandit models (Holland 1969). He used these ideas to develop a more theoretical analysis of his *reproductive plans* (simple GAs) (Holland 1971, 1973). Holland then pulled all of these ideas together in his pivotal book *Adaptation in Natural and Artificial Systems* (Holland 1975). B2.5

Of interest was the fact that many of the desirable properties of these algorithms being identified by Holland theoretically were frequently not observed experimentally. It was not difficult to identify the reasons for this. Hampered by a lack of computational resources and analysis tools, most of the early experimental studies involved a relatively small number of runs using small population sizes (generally

less than 20). It became increasingly clear that many of the observed deviations from expected behavior could be traced to the well-known phenomenon in population genetics of *genetic drift*, the loss of genetic diversity due to the stochastic aspects of selection, reproduction, and the like in small populations.

By the early 1970s there was considerable interest in understanding better the behavior of implementable GAs. In particular, it was clear that choices of population size, representation issues, the choice of operators and operator rates all had significant effects of the observed behavior of GAs. Frantz's thesis (Frantz 1972) reflected this new focus by studying in detail the roles of crossover and inversion in populations of size 100. Of interest here is some of the earliest experimental work on multipoint crossover operators.

De Jong's thesis (De Jong 1975) broaded this line of study by analyzing both theoretically and experimentally the interacting effects of population size, crossover, and mutation on the behavior of a family of GAs being used to optimize a fixed test suite of functions. Out of this study came a strong sense that even these simple GAs had significant potential for solving difficult optimization problems.

The mid-1970s also represented a branching out of the family tree of GAs as other universities and research laboratories established research activities in this area. This happened slowly at first since initial attempts to spread the word about the progress being made in GAs were met with fairly negative perceptions from the artificial intelligence (AI) community as a result of early overhyped work in areas such as self-organizing systems and perceptrons.

Undaunted, groups from several universities including the University of Michigan, the University of Pittsburgh, and the University of Alberta organized an *Adaptive Systems Workshop* in the summer of 1976 in Ann Arbor, Michigan. About 20 people attended and agreed to meet again the following summer. This pattern repeated itself for several years, but by 1979 the organizers felt the need to broaden the scope and make things a little more formal. Holland, De Jong, and Sampson obtained NSF funding for *An Interdisciplinary Workshop in Adaptive Systems*, which was held at the University of Michigan in the summer of 1981 (Sampson 1981).

By this time there were several established research groups working on GAs. At the University of Michigan, Bethke, Goldberg, and Booker were continuing to develop GAs and explore Holland's *classifier systems* as part of their PhD research (Bethke 1981, Booker 1982, Goldberg 1983). At the University of Pittsburgh, Smith and Wetzel were working with De Jong on various GA enhancements including the *Pitt approach* to rule learning (Smith 1980, Wetzel 1983). At the University of Alberta, Brindle continued to look at optimization applications of GAs under the direction of Sampson (Brindle 1981). B1.5.2

The continued growth of interest in GAs led to a series of discussions and plans to hold the first *International Conference on Genetic Algorithms (ICGA)* in Pittsburgh, Pennsylvania, in 1985. There were about 75 participants presenting and discussing a wide range of new developments in both the theory and application of GAs (Grefenstette 1985). The overwhelming success of this meeting resulted in agreement to continue *ICGA* as a biannual conference. Also agreed upon at *ICGA'85* was the initiation of a moderated electronic discussion group called *GA List*.

The field continued to grow and mature as reflected by the *ICGA* conference activities (Grefenstette 1987, Schaffer 1989) and the appearance of several books on the subject (Davis 1987, Goldberg 1989). Goldberg's book, in particular, served as a significant catalyst by presenting current GA theory and applications in a clear and precise form easily understood by a broad audience of scientists and engineers.

By 1989 the *ICGA* conference and other GA-related activities had grown to a point that some more formal mechanisms were needed. The result was the formation of the International Society for Genetic Algorithms (ISGA), an incorporated body whose purpose is to serve as a vehicle for conference funding and to help coordinate and facilitate GA-related activities. One of its first acts of business was to support a proposal to hold a theory workshop on the *Foundations of Genetic Algorithms (FOGA)* in Bloomington, Indiana (Rawlins 1991).

By this time nonstandard GAs were being developed to evolve complex, nonlinear variable-length structures such as rule sets, LISP code, and neural networks. One of the motivations for *FOGA* was the sense that the growth of GA-based applications had driven the field well beyond the capacity of existing theory to provide effective analyses and predictions.

Also in 1990, Schwefel hosted the first *PPSN* conference in Dortmund, which resulted in the first organized interaction between the ES and GA communities. This led to additional interaction at *ICGA'91* in San Diego which resulted in an informal agreement to hold *ICGA* and *PPSN* in alternating years, and a commitment to jointly initiate a journal for the field.

It was felt that in order for the journal to be successful, it must have broad scope and include other species of EA. Efforts were made to include the EP community as well (which began to organize its own conferences in 1992), and the new journal *Evolutionary Computation* was born with the inaugural issue in the spring of 1993.

The period from 1990 to the present has been characterized by tremendous growth and diversity of the GA community as reflected by the many conference activities (e.g. *ICGA* and *FOGA*), the emergence of new books on GAs, and a growing list of journal papers. New paradigms such as *messy GAs* (Goldberg *et al* 1991) and *genetic programming* (Koza 1992) were being developed. The interactions with other EC communities resulted in considerable crossbreeding of ideas and many new hybrid EAs. New GA applications continue to be developed, spanning a wide range of problem areas from engineering design problems to operations research problems to automatic programming.

C4.2.4 B1.5.1

A2.3.4 Evolution strategies

In 1964, three students of the Technical University of Berlin, Bienert, Rechenberg, and Schwefel, did not at all aim at devising a new kind of optimization procedure. During their studies of aerotechnology and space technology they met at an Institute of Fluid Mechanics and wanted to construct a kind of research robot that should perform series of experiments on a flexible slender three-dimensional body in a wind tunnel so as to minimize its drag. The method of minimization was planned to be either a one variable at a time or a discrete gradient technique, gleaned from classical numerics. Both strategies, performed manually, failed, however. They became stuck prematurely when used for a two-dimensional demonstration facility, a joint plate—its optimal shape being a flat plate—with which the students tried to demonstrate that it was possible to find the optimum automatically.

Only then did Rechenberg (1965) hit upon the idea to use dice for random decisions. This was the breakthrough—on 12 June 1964. The first version of an evolutionary strategy (ES), later called the $(1 + 1)$ ES, was born, with discrete, binomially distributed mutations centered at the ancestor's position, and just one parent and one descendant per generation. This ES was first tested on a mechanical calculating machine by Schwefel before it was used for the *experimentum crucis*, the joint plate. Even then, it took a while to overcome a merely locally optimal S shape and to converge towards the expected global optimum, the flat plate. Bienert (1967), the third of the three students, later actually constructed a kind of robot that could perform the actions and decisions automatically.

Using this simple *two-membered* ES, another student, Lichtfuß (1965), optimized the shape of a bent pipe, also experimentally. The result was rather unexpected, but nevertheless obviously better than all shapes proposed so far.

First computer experiments, on a Zuse Z23, as well as analytical investigations using binomially distributed integer mutations, had already been performed by Schwefel (1965). The main result was that such a strategy can become stuck prematurely, i.e. at 'solutions' that are not even locally optimal. Based on this experience the use of normally instead of binomially distributed mutations became standard in most of the later computer experiments with real-valued variables and in theoretical investigations into the method's efficiency, but not however in experimental optimization using ESs. In 1966 the little ES community was destroyed by dismissal from the Institute of Fluid Mechanics ('Cybernetics as such is no longer pursued at the institute!'). Not before 1970 was it found together again at the Institute of Measurement and Control of the Technical University of Berlin, sponsored by grants from the German Research Foundation (DFG). Due to the circumstances, the group missed publishing its ideas and results properly, especially in English.

In the meantime the often-cited two-phase nozzle optimization was performed at the Institute of Nuclear Technology of the Technical University of Berlin, then in an industrial surrounding, the AEG research laboratory (Schwefel 1968, Klockgether and Schwefel 1970), also at Berlin. For a hot-water flashing flow the shape of a three-dimensional convergent–divergent (thus supersonic) nozzle with maximum energy efficiency was sought. Though in this experimental optimization an exogenously controlled binomial-like distribution was used again, it was the first time that gene duplication and deletion were incorporated into an EA, especially in a $(1 + 1)$ ES, because the optimal length of the nozzle was not known in advance. As in case of the bent pipe this experimental strategy led to highly unexpected results, not easy to understand even afterwards, but definitely much better than available before.

First Rechenberg and later Schwefel analyzed and improved their ES. For the $(1+1)$ ES, Rechenberg, in his Dr.-Ing. thesis of 1971, developed, on the basis of two convex n-dimensional model functions, a

convergence rate theory for $n \gg 1$ variables. Based on these results he formulated a $\frac{1}{5}$ success rule for adapting the standard deviation of mutation (Rechenberg 1973). The hope of arriving at an even better strategy by imitating organic evolution more closely led to the incorporation of the population principle and the introduction of recombination, which of course could not be embedded in the $(1 + 1)$ ES. A first *multimembered* ES, the $(\mu + 1)$ ES—the notation was introduced later by Schwefel—was also designed by Rechenberg in his seminal work of 1973. Because of its inability to self-adapt the mutation step sizes (more accurately, standard deviations of the mutations), this strategy was never widely used.

Much more widespread became the $(\mu+\lambda)$ ES and (μ, λ) ES, both formulated by Schwefel in his Dr.-Ing. thesis of 1974–1975. It contains theoretical results such as a convergence rate theory for the $(1+\lambda)$ ES and the $(1, \lambda)$ ES ($\lambda > 1$), analogous to the theory introduced by Rechenberg for the $(1+1)$ ES (Schwefel 1977). The *multimembered* ($\mu > 1$) ESs arose from the otherwise ineffective incorporation of mutatable mutation parameters (variances and covariances of the Gaussian distributions used). Self-adaptation was achieved with the (μ, λ) ES first, not only with respect to the step sizes, but also with respect to correlation coefficients. The enhanced ES version with correlated mutations, described already in an internal report (Schwefel 1974), was published much later (Schwefel 1981) due to the fact that the author left Berlin in 1976. A more detailed empirical analysis of the on-line self-adaptation of the internal or strategy parameters was first published by Schwefel in 1987 (the tests themselves were secretly performed on one of the first small instruction multiple data (SIMD) parallel machines (CRAY1) at the Nuclear Research Centre (KFA) Jülich during the early 1980s with a first parallel version of the multimembered ES with correlated mutations). It was in this work that the notion of *self-adaptation by collective learning* first came up. The importance of recombination (for object as well as strategy parameters) and soft selection (or $\mu > 1$) was clearly demonstrated. Only recently has Beyer (1995a, b) delivered the theoretical background to that particularly important issue.

It may be worth mentioning that in the beginning there were strong objections against increasing λ as well as μ beyond one. The argument against $\lambda > 1$ was that the exploitation of the current knowledge was unnecessarily delayed, and the argument against $\mu > 1$ was that the survival of inferior members of the population would unnecessarily slow down the evolutionary progress. The hint that λ successors could be evaluated in parallel did not convince anybody since parallel computers were neither available nor expected in the near future. The two-membered ES and the very similar creeping random search method of Rastrigin (1965) were investigated thoroughly with respect to their convergence and convergence rates also by Matyas (1965) in Czechoslovakia, Born (1978) on the Eastern side of the Berlin wall (!), and Rappl (1984) in Munich.

Since this early work many new results have been produced by the ES community consisting of the group at Berlin (Rechenberg, since 1972) and that at Dortmund (Schwefel, since 1985). In particular, strategy variants concerning other than only real-valued parameter optimization, i.e. real-world problems, were invented. The first use of an ES for binary optimization using multicellular individuals was presented by Schwefel (1975). The idea of using several subpopulations and *niching mechanisms* for global optimization was propagated by Schwefel in 1977; due to a lack of computing resources, however, it could not be tested thoroughly at that time. Rechenberg (1978) invented a notational scheme for such nested ESs. C6.1

Beside these nonstandard approaches there now exists a wide range of other ESs, e.g. several parallel concepts (Hoffmeister and Schwefel 1990, Lohmann 1991, Rudolph 1991, 1992, Sprave 1994, Rudolph and Sprave 1995), ESs for *multicriterion problems* (Kursawe 1991, 1992), for mixed-integer tasks (Lohmann 1992, Rudolph 1994, Bäck and Schütz 1995), and even for problems with a variable-dimensional parameter space (Schütz and Sprave 1996), and variants concerning nonstandard step size and direction adaptation schemes (see e.g. Matyas 1967, Stewart *et al* 1967, Fürst *et al* 1968, Heydt 1970, Rappl 1984, Ostermeier *et al* 1994). Comparisons between ESs, GAs, and EP may be found in the articles by Bäck *et al* (1991, 1993). It was Bäck (1996) who introduced a common algorithmic scheme for all brands of current EAs. F1.9

Omitting all these other useful nonstandard ESs—a commented collection of literature concerning ES applications was made at the University of Dortmund (Bäck *et al* 1992)—the history of ESs is closed with a mention of three recent books by Rechenberg (1994), Schwefel (1995), and Bäck (1996) as well as three recent contributions that may be seen as written tutorials (Schwefel and Rudolph 1995, Bäck and Schwefel 1995, Schwefel and Bäck 1995), which on the one hand define the actual standard ES algorithms and on the other hand present some recent theoretical results.

References

Altenberg L 1994 Emergent phenomena in genetic programming *Proc. 3rd Annu. Conf. on Evolutionary Programming (San Diego, CA, 1994)* ed A V Sebald and L J Fogel (Singapore: World Scientific) pp 233–41

Andersen B, McDonnell J and Page W 1992 Configuration optimization of mobile manipulators with equality constraints using evolutionary programming *Proc. 1st Ann. Conf. on Evolutionary Programming (La Jolla, CA, 1992)* ed D B Fogel and W Atmar (La Jolla, CA: Evolutionary Programming Society) pp 71–9

Angeline P J, Fogel D B and Fogel L J 1996 A comparison of self-adaptation methods for finite state machines in a dynamic environment *Evolutionary Programming V—Proc. 5th Ann. Conf. on Evolutionary Programming (1996)* ed L J Fogel, P J Angeline and T Bäck (Cambridge, MA: MIT Press)

Angeline P J, Saunders G M and Pollack J B 1994 An evolutionary algorithm that constructs recurrent neural networks *IEEE Trans. Neural Networks* **NN-5** 54–65

Atmar J W 1976 *Speculation of the Evolution of Intelligence and Its Possible Realization in Machine Form* ScD Thesis, New Mexico State University

Bäck T 1996 *Evolutionary Algorithms in Theory and Practice* (New York: Oxford University Press)

Bäck T, Hoffmeister F and Schwefel H-P 1991 A survey of evolution strategies *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, 1991)* ed R K Belew and L B Booker (San Mateo, CA: Morgan Kaufmann) pp 2–9

——1992 *Applications of Evolutionary Algorithms* Technical Report of the University of Dortmund Department of Computer Science Systems Analysis Research Group SYS-2/92

Bäck T, Rudolph G and Schwefel H-P 1993 Evolutionary programming and evolution strategies: similarities and differences *Proc. 2nd Ann. Conf. on Evolutionary Programming (San Diego, CA, 1993)* ed D B Fogel and W Atmar (La Jolla, CA: Evolutionary Programming Society) pp 11–22

Bäck T and Schütz M 1995 Evolution strategies for mixed-integer optimization of optical multilayer systems *Evolutionary Programming IV—Proc. 4th Ann. Conf on Evolutionary Programming (San Diego, CA, 1995)* ed J R McDonnell, R G Reynolds and D B Fogel (Cambridge, MA: MIT Press) pp 33–51

Bäck T and Schwefel H-P 1995 Evolution strategies I: variants and their computational implementation *Genetic Algorithms in Engineering and Computer Science, Proc. 1st Short Course EUROGEN-95* ed G Winter, J Périaux, M Galán and P Cuesta (New York: Wiley) pp 111–26

Bagley J D 1967 *The Behavior of Adaptive Systems which Employ Genetic and Correlation Algorithms* PhD Thesis, University of Michigan

Belew R K and Booker L B (eds) 1991 *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, 1991)* (San Mateo, CA: Morgan Kaufmann)

Bethke A D 1981 *Genetic Algorithms as Function Optimizers* PhD Thesis, University of Michigan

Beyer H-G 1995a *How GAs do Not Work—Understanding GAs Without Schemata and Building Blocks* Technical Report of the University of Dortmund Department of Computer Science Systems Analysis Research Group SYS-2/95

——1995b Toward a theory of evolution strategies: on the benefit of sex—the $(\mu/\mu, \lambda)$-theory *Evolutionary Comput.* **3** 81–111

Bhattacharjya A K and Roysam B 1994 Joint solution of low-, intermediate- and high-level vision tasks by evolutionary optimization: application to computer vision at low SNR *IEEE Trans. Neural Networks* **NN-5** 83–95

Bienert P 1967 *Aufbau einer Optimierungsautomatik für drei Parameter* Dipl.-Ing. Thesis, Technical University of Berlin, Institute of Measurement and Control Technology

Booker L 1982 *Intelligent Behavior as an Adaptation to the Task Environment* PhD Thesis, University of Michigan

Born J 1978 *Evolutionsstrategien zur numerischen Lösung von Adaptationsaufgaben* PhD Thesis, Humboldt University at Berlin

Box G E P 1957 Evolutionary operation: a method for increasing industrial productivity *Appl. Stat.* **6** 81–101

Box G E P and Draper N P 1969 *Evolutionary Operation. A Method for Increasing Industrial Productivity* (New York: Wiley)

Bremermann H J 1962 Optimization through evolution and recombination *Self-Organizing Systems* ed M C Yovits *et al* (Washington, DC: Spartan)

Bremermann H J, Rogson M and Salaff S 1965 Search by evolution *Biophysics and Cybernetic Systems—Proc. 2nd Cybernetic Sciences Symp.* ed M Maxfield, A Callahan and L J Fogel (Washington, DC: Spartan) pp 157–67

Brindle A 1981 *Genetic Algorithms for Function Optimization* PhD Thesis, University of Alberta

Brotherton T W, Simpson P K, Fogel D B and Pollard T 1994 Classifier design using evolutionary programming *Proc. 3rd Ann. Conf. on Evolutionary Programming (San Diego, CA, 1994)* ed A V Sebald and L J Fogel (Singapore: World Scientific) pp 68–75

Burgin G H 1969 On playing two-person zero-sum games against nonminimax players *IEEE Trans. Syst. Sci. Cybernet.* **SSC-5** 369–70

Cavicchio D J 1970 *Adaptive Search Using Simulated Evolution* PhD Thesis, University of Michigan

Davis L 1987 *Genetic Algorithms and Simulated Annealing* (London: Pitman)

Dearholt D W 1976 Some experiments on generalization using evolving automata *Proc. 9th Int. Conf. on System Sciences (Honolulu, HI)* pp 131–3

De Jong K A 1975 *Analysis of Behavior of a Class of Genetic Adaptive Systems* PhD Thesis, University of Michigan

English T M 1994 Generalization in populations of recurrent neural networks *Proc. 3rd Ann. Conf. on Evolutionary Programming (San Diego, CA, 1994)* ed A V Sebald and L J Fogel (Singapore: World Scientific) pp 26–33

Fogel D B 1988 An evolutionary approach to the traveling salesman problem *Biol. Cybernet.* **60** 139–44

——1989 Evolutionary programming for voice feature analysis *Proc. 23rd Asilomar Conf. on Signals, Systems and Computers (Pacific Grove, CA)* pp 381–3

——1995 *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence* (New York: IEEE)

Fogel D B and Atmar J W 1990 Comparing genetic operators with Gaussian mutations in simulated evolutionary processing using linear systems *Biol. Cybernet.* **63** 111–4

——(eds) 1992 *Proc. 1st Ann. Conf. on Evolutionary Programming (La Jolla, CA, 1992)* (La Jolla, CA: Evolutionary Programming Society)

Fogel D B and Fogel L J 1988 Route optimization through evolutionary programming *Proc. 22nd Asilomar Conf. on Signals, Systems and Computers (Pacific Grove, CA)* pp 679–80

Fogel D B, Fogel L J and Atmar J W 1991 Meta-evolutionary programming *Proc. 25th Asilomar Conf. on Signals, Systems and Computers (Pacific Grove, CA)* ed R R Chen pp 540–5

Fogel D B, Fogel L J, Atmar J W and Fogel G B 1992 Hierarchic methods of evolutionary programming *Proc. 1st Ann. Conf. on Evolutionary Programming (La Jolla, CA, 1992)* ed D B Fogel and W Atmar (La Jolla, CA: Evolutionary Programming Society) pp 175–82

Fogel D B, Fogel L J and Porto V W 1990 Evolving neural networks *Biol. Cybernet.* **63** 487–93

Fogel D B, Wasson E C and Boughton E M 1995 Evolving neural networks for detecting breast cancer *Cancer Lett.* **96** 49–53

Fogel L J 1962 Autonomous automata *Industrial Res.* **4** 14–9

——1963 *Biotechnology: Concepts and Applications* (Englewood Cliffs, NJ: Prentice-Hall)

——1964 *On the Organization of Intellect* PhD Thesis, University of California at Los Angeles

——1968 Extending communication and control through simulated evolution *Bioengineering—an Engineering View Proc. Symp. on Engineering Significance of the Biological Sciences* ed G Bugliarello (San Francisco, CA: San Francisco Press) pp 286–304

Fogel L J, Angeline P J and Bäck T (eds) 1996 *Evolutionary Programming V—Proc. 5th Ann. Conf. on Evolutionary Programming (1996)* (Cambridge, MA: MIT Press)

Fogel L J and Burgin G H 1969 *Competitive Goal-seeking through Evolutionary Programming* Air Force Cambridge Research Laboratories Final Report Contract AF 19(628)-5927

Fogel L J and Fogel D B 1986 *Artificial Intelligence through Evolutionary Programming* US Army Research Institute Final Report Contract PO-9-X56-1102C-1

Fogel L J, Owens A J and Walsh M J 1964 On the evolution of artificial intelligence *Proc. 5th Natl Symp. on Human Factors in Electronics* (San Diego, CA: IEEE)

——1965 Artificial intelligence through a simulation of evolution *Biophysics and Cybernetic Systems* ed A Callahan, M Maxfield and L J Fogel (Washington, DC: Spartan) pp 131–56

——1966 *Artificial Intelligence through Simulated Evolution* (New York: Wiley)

Frantz D R 1972 *Non-linearities in Genetic Adaptive Search* PhD Thesis, University of Michigan

Fraser A S 1957 Simulation of genetic systems by automatic digital computers *Aust. J. Biol. Sci.* **10** 484–99

Friedberg R M 1958 A learning machine: part I *IBM J.* **2** 2–13

Friedberg R M, Dunham B and North J H 1959 A learning machine: part II *IBM J.* **3** 282–7

Fürst H, Müller P H and Nollau V 1968 Eine stochastische Methode zur Ermittlung der Maximalstelle einer Funktion von mehreren Veränderlichen mit experimentell ermittelbaren Funktionswerten und ihre Anwendung bei chemischen Prozessen *Chem.–Tech.* **20** 400–5

Gehlhaar *et al* 1995Gehlhaar D K *et al* 1995 Molecular recognition of the inhibitor AG-1343 by HIV-1 protease: conformationally flexible docking by evolutionary programming *Chem. Biol.* **2** 317–24

Gell-Mann M 1994 *The Quark and the Jaguar* (New York: Freeman)

Goldberg D E 1983 *Computer-Aided Gas Pipeline Operation using Genetic Algorithms and Rule Learning* PhD Thesis, University of Michigan

——1989 *Genetic Algorithms in Search, Optimization and Machine Learning* (Reading, MA: Addison-Wesley)

Goldberg D E, Deb K and Korb B 1991 Don't worry, be messy *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, 1991)* ed R K Belew and L B Booker (San Mateo, CA: Morgan Kaufmann) pp 24–30

Grefenstette J J (ed) 1985 *Proc. 1st Int. Conf. on Genetic Algorithms and Their Applications (Pittsburgh, PA, 1985)* (Hillsdale, NJ: Erlbaum)

——1987 *Proc. 2nd Int. Conf. on Genetic Algorithms and Their Applications (Cambridge, MA, 1987)* (Hillsdale, NJ: Erlbaum)

Haffner S B and Sebald A V 1993 Computer-aided design of fuzzy HVAC controllers using evolutionary programming *Proc. 2nd Ann. Conf. on Evolutionary Programming (San Diego, CA, 1993)* ed D B Fogel and W Atmar (La Jolla, CA: Evolutionary Programming Society) pp 98–107

Heydt G T 1970 *Directed Random Search* PhD Thesis, Purdue University

Hoffmeister F and Schwefel H-P 1990 A taxonomy of parallel evolutionary algorithms *Parcella '90, Proc. 5th Int. Workshop on Parallel Processing by Cellular Automata and Arrays* vol 2, ed G Wolf, T Legendi and U Schendel (Berlin: Academic) pp 97–107

Holland J H 1962 Outline for a logical theory of adaptive systems *J. ACM* **9** 297–314

——1967 Nonlinear environments permitting efficient adaptation *Computer and Information Sciences II* (New York: Academic)

——1969 Adaptive plans optimal for payoff-only environments *Proc. 2nd Hawaii Int. Conf. on System Sciences* pp 917–20

——1971 Processing and processors for schemata *Associative information processing* ed E L Jacks (New York: Elsevier) pp 127–46

——1973 Genetic algorithms and the optimal allocation of trials *SIAM J. Comput.* **2** 88–105

——1975 *Adaptation in Natural and Artificial Systems* (Ann Arbor, MI: University of Michigan Press)

Hollstien R B 1971 *Artificial Genetic Adaptation in Computer Control Systems* PhD Thesis, University of Michigan

Kim J-H and Jeon J-Y 1996 Evolutionary programming-based high-precision controller design *Evolutionary Programming V—Proc. 5th Ann. Conf. on Evolutionary Programming (1996)* ed L J Fogel, P J Angeline and T Bäck (Cambridge, MA: MIT Press)

Klockgether J and Schwefel H-P 1970 Two-phase nozzle and hollow core jet experiments *Proc. 11th Symp. on Engineering Aspects of Magnetohydrodynamics* ed D G Elliott (Pasadena, CA: California Institute of Technology) pp 141–8

Koza J R 1992 *Genetic Programming* (Cambridge, MA: MIT Press)

Koza J R and Andre D 1996 Evolution of iteration in genetic programming *Evolutionary Programming V—Proc. 5th Ann. Conf. on Evolutionary Programming (1996)* ed L J Fogel, P J Angeline and T Bäck (Cambridge, MA: MIT Press)

Kursawe F 1991 A variant of evolution strategies for vector optimization *Parallel Problem Solving from Nature—Proc. 1st Workshop PPSN I (Lecture Notes in Computer Science 496) (Dortmund, 1991)* ed H-P Schwefel and R Männer (Berlin: Springer) pp 193–7

——1992 Naturanaloge Optimierverfahren—Neuere Entwicklungen in der Informatik *Studien zur Evolutorischen Ökonomik II (Schriften des Vereins für Socialpolitik 195 II)* ed U Witt (Berlin: Duncker and Humblot) pp 11–38

Land M and Belew R K 1995 Towards a self-replicating language for computation *Evolutionary Programming IV—Proc. 4th Ann. Conf on Evolutionary Programming (San Diego, CA, 1995)* ed J R McDonnell, R G Reynolds and D B Fogel (Cambridge, MA: MIT Press) pp 403–13

Larsen R W and Herman J S 1992 A comparison of evolutionary programming to neural networks and an application of evolutionary programming to a navy mission planning problem *Proc. 1st Ann. Conf. on Evolutionary Programming (La Jolla, CA, 1992)* ed D B Fogel and W Atmar (La Jolla, CA: Evolutionary Programming Society) pp 127–33

Lichtfuß H J 1965 *Evolution eines Rohrkrümmers* Dipl.-Ing. Thesis, Technical University of Berlin, Hermann Föttinger Institute for Hydrodynamics

Lohmann R 1991 Application of evolution strategy in parallel populations *Parallel Problem Solving from Nature—Proc. 1st Workshop PPSN I (Dortmund, 1991) (Lecture Notes in Computer Science 496)* ed H-P Schwefel and R Männer (Berlin: Springer) pp 198–208

——1992 Structure evolution and incomplete induction *Parallel Problem Solving from Nature 2 (Brussels, 1992)* ed R Männer and B Manderick (Amsterdam: Elsevier–North-Holland) pp 175–85

Lutter B E and Huntsinger R C 1969 Engineering applications of finite automata *Simulation* **13** 5–11

Männer R and Manderick B (eds) 1992 *Parallel Problem Solving from Nature 2 (Brussels, 1992)* (Amsterdam: Elsevier–North-Holland)

Matyas J 1965 Random optimization *Automation Remote Control* **26** 244–51

——1967 Das zufällige Optimierungsverfahren und seine Konvergenz *Proc. 5th Int. Analogue Computation Meeting (Lausanne, 1967)* **1** 540–4

McDonnell J R 1992 Training neural networks with weight constraints *Proc. 1st Ann. Conf. on Evolutionary Programming (La Jolla, CA, 1992)* ed D B Fogel and W Atmar (La Jolla, CA: Evolutionary Programming Society) pp 111–9

McDonnell J R, Andersen B D, Page W C and Pin F 1992 Mobile manipulator configuration optimization using evolutionary programming *Proc. 1st Ann. Conf. on Evolutionary Programming (La Jolla, CA, 1992)* ed D B Fogel and W Atmar (La Jolla, CA: Evolutionary Programming Society) pp 52–62

McDonnell J R and Waagen D 1994 Evolving recurrent perceptrons for time-series prediction *IEEE Trans. Neural Networks* **NN-5** 24–38

Michalewicz Z *et al* (eds) 1994 *Proc. 1st IEEE Conf. on Evolutionary Computation (Orlando, FL, 1994)* (Piscataway, NJ: IEEE)

Ostermeier A, Gawelczyk A and Hansen N 1994 Step-size adaptation based on non-local use of selection information *Parallel Problem Solving from Nature—PPSN III Int. Conf. on Evolutionary Computation (Jerusalem, 1994) (Lecture notes in Computer Science 866)* ed Y Davidor, H-P Schwefel and R Männer (Berlin: Springer) pp 189–98

Page W C, Andersen B D and McDonnell J R 1992 An evolutionary programming approach to multi-dimensional path planning *Proc. 1st Ann. Conf. on Evolutionary Programming (La Jolla, CA, 1992)* ed D B Fogel and W Atmar (La Jolla, CA: Evolutionary Programming Society) pp 63–70

Porto V W 1992 Alternative methods for training neural networks *Proc. 1st Ann. Conf. on Evolutionary Programming (La Jolla, CA, 1992)* ed D B Fogel and W Atmar (La Jolla, CA: Evolutionary Programming Society) pp 100–10

Porto V W, Fogel D B and Fogel L J 1995 Alternative neural network training methods *IEEE Expert* **10** 16–22

Rappl G 1984 *Konvergenzraten von Random-Search-Verfahren zur globalen Optimierung* PhD Thesis, Bundeswehr University

Rastrigin L A 1965 *Random Search in Optimization Problems for Multiparameter Systems (translated from the Russian original: Sluchainyi poisk v zadachakh optimisatsii mnogoarametricheskikh sistem, Zinatne, Riga)* Air Force System Command Foreign Technology Division FTD-HT-67-363

Rawlins G J E (ed) 1991 *Foundations of Genetic Algorithms* (San Mateo, CA: Morgan Kaufmann)

Rechenberg I 1965 *Cybernetic Solution Path of an Experimental Problem* Royal Aircraft Establishment Library Translation 1122

——1973 *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution* (Stuttgart: Frommann–Holzboog)

——1978 Evolutionsstrategien *Simulationsmethoden in der Medizin und Biologie* ed B Schneider and U Ranft (Berlin: Springer) pp 83–114

——1994 *Evolutionsstrategie '94* (Stuttgart: Frommann–Holzboog)

Rizki M M, Tamburino L A and Zmuda M A 1995 Evolution of morphological recognition systems *Evolutionary Programming IV—Proc. 4th Ann. Conf on Evolutionary Programming (San Diego, CA, 1995)* ed J R McDonnell, R G Reynolds and D B Fogel (Cambridge, MA: MIT Press) pp 95–106

Rosenberg R 1967 *Simulation of Genetic Populations with Biochemical Properties* PhD Thesis, University of Michigan

Rudolph G 1991 Global optimization by means of distributed evolution strategies *Parallel Problem Solving from Nature—Proc. 1st Workshop PPSN I (Dortmund, 1991) (Lecture Notes in Computer Science 496)* ed H-P Schwefel and R Männer (Berlin: Springer) pp 209–13

——1992 Parallel approaches to stochastic global optimization *Parallel Computing: from Theory to Sound Practice, Proc. Eur. Workshop on Parallel Computing* ed W Joosen and E Milgrom (Amsterdam: IOS) pp 256–67

——1994 An evolutionary algorithm for integer programming *Parallel Problem Solving from Nature—PPSN III Int. Conf. on Evolutionary Computation (Jerusalem, 1994) (Lecture notes in Computer Science 866)* ed Y Davidor, H-P Schwefel and R Männer (Berlin: Springer) pp 139–48

Rudolph G and Sprave J 1995 A cellular genetic algorithm with self-adjusting acceptance threshold *Proc. 1st IEE/IEEE Int. Conf. on Genetic Algorithms in Engineering Systems: Innovations and Applications (GALESIA '95) (Sheffield, 1995)* (London: IEE) pp 365–72

Sampson J R 1981 *A Synopsis of the Fifth Annual Ann Arbor Adaptive Systems Workshop* Department of Computing and Communication Science, Logic of Computers Group Technical Report University of Michigan

Saravanan N, Fogel D B and Nelson K M 1995 A comparison of methods for self-adaptation in evolutionary algorithms *BioSystems* **36** 157–66

Satterthwaite F E 1959a Random balance experimentation *Technometrics* **1** 111–37

——1959b *REVOP or Random Evolutionary Operation* Merrimack College Technical Report 10-10-59

Schaffer J D (ed) 1989 *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, WA, 1989)* (San Mateo, CA: Morgan Kaufmann)

Schütz M and Sprave J 1996 Application of parallel mixed-integer evolution strategies with mutation rate pooling *Evolutionary Programming V—Proc. 5th Ann. Conf. on Evolutionary Programming (1996)* ed L J Fogel, P J Angeline and T Bäck (Cambridge, MA: MIT Press)

Schwefel H-P 1965 *Kybernetische Evolution als Strategie der experimentellen Forschung in der Strömungstechnik* Dipl.-Ing. Thesis, Technical University of Berlin, Hermann Föttinger Institute for Hydrodynamics

——1968 *Experimentelle Optimierung einer Zweiphasendüse Teil I* AEG Research Institute Project MHD-Staustrahlrohr 11034/68 Technical Report 35

——1974 *Adaptive Mechanismen in der biologischen Evolution und ihr Einflußauf die Evolutionsgeschwindigkeit* Technical University of Berlin Working Group of Bionics and Evolution Techniques at the Institute for Measurement and Control Technology Technical Report Re 215/3

——1975 *Binäre Optimierung durch somatische Mutation* Working Group of Bionics and Evolution Techniques at the Institute of Measurement and Control Technology of the Technical University of Berlin and the Central Animal Laboratory of the Medical Highschool of Hannover Technical Report

——1977 *Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie (Interdisciplinary Systems Research 26)* (Basle: Birkhäuser)

——1981 *Numerical Optimization of Computer Models* (Chichester: Wiley)

——1987 Collective phenomena in evolutionary systems *Problems of Constancy and Change—the Complementarity of Systems Approaches to Complexity, Papers Presented at the 31st Ann. Meeting Int. Society Gen. Syst. Res.* vol 2, ed P Checkland and I Kiss (Budapest: International Society for General System Research) pp 1025–33

——1995 *Evolution and Optimum Seeking* (New York: Wiley)

Schwefel H-P and Bäck T 1995 Evolution strategies II: theoretical aspects *Genetic Algorithms in Engineering and Computer Science Proc. 1st Short Course EUROGEN-95* ed G Winter, J Périaux, M Galán and P Cuesta (New York: Wiley) pp 127–40

Schwefel H-P and Männer R (eds) 1991 *Parallel Problem Solving from Nature—Proc. 1st Workshop PPSN I (Dortmund, 1991) (Lecture Notes in Computer Science 496)* (Berlin: Springer)

Schwefel H-P and Rudolph G 1995 Contemporary evolution strategies *Advances in Artificial Life—Proc. 3rd Eur. Conf. on Artificial Life (ECAL'95) (Lecture Notes in Computer Science 929)* ed F Morán, A Moreno, J J Merelo and P Chacón (Berlin: Springer) pp 893–907

Sebald A V and Fogel D B 1992 Design of fault-tolerant neural networks for pattern classification *Proc. 1st Ann. Conf. on Evolutionary Programming (La Jolla, CA, 1992)* ed D B Fogel and W Atmar (La Jolla, CA: Evolutionary Programming Society) pp 90–9

Sebald A V, Schlenzig J and Fogel D B 1992 Minimax design of CMAC encoded neural controllers for systems with variable time delay *Proc. 1st Ann. Conf. on Evolutionary Programming (La Jolla, CA, 1992)* ed D B Fogel and W Atmar (La Jolla, CA: Evolutionary Programming Society) pp 120–6

Smith S F 1980 *A Learning System Based on Genetic Adaptive Algorithms* PhD Thesis, University of Pittsburgh

Spendley W, Hext G R and Himsworth F R 1962 Sequential application of simplex designs in optimisation and evolutionary operation *Technometrics* **4** 441–61

Sprave J 1994 Linear neighborhood evolution strategy *Proc. 3rd Ann. Conf. on Evolutionary Programming (San Diego, CA, 1994)* ed A V Sebald and L J Fogel (Singapore: World Scientific) pp 42–51

Stewart E C, Kavanaugh W P and Brocker D H 1967 Study of a global search algorithm for optimal control *Proc. 5th Int. Analogue Computation Meeting (Lausanne, 1967)* vol 1, pp 207–30

Takeuchi A 1980 Evolutionary automata—comparison of automaton behavior and Restle's learning model *Information Sci.* **20** 91–9

Wetzel A 1983 *Evaluation of the Effectiveness of Genetic Algorithms in Combinatorial Optimization* unpublished manuscript, University of Pittsburgh

B1.1 Introduction

Thomas Bäck

Abstract

Within this introduction to Chapter B1, we present a general outline of an evolutionary algorithm, which is consistent with all mainstream instances of evolutionary computation and summarizes the major features common to evolutionary computation approaches; that is, a population of individuals, recombination and/or mutation, and selection. Some of the differences between genetic algorithms, evolution strategies, and evolutionary programming are briefly mentioned to provide a short overview of the features that are most emphasized by these different approaches. Furthermore, the basic characteristics of genetic programming and classifier systems are also outlined.

B1.1.1 General outline of evolutionary algorithms

Since they are gleaned from the model of organic evolution, all basic instances of evolutionary algorithms share a number of common properties, which are mentioned here to characterize the prototype of a general evolutionary algorithm:

(i) Evolutionary algorithms utilize the collective learning process of a population of individuals. Usually, each individual represents (or encodes) a search point in the space of potential solutions to a given problem. Additionally, individuals may also incorporate further information; for example, *strategy parameters* of the evolutionary algorithm. C1.3.2, C3.2.2

(ii) Descendants of individuals are generated by randomized processes intended to model *mutation* and *recombination*. Mutation corresponds to an erroneous self-replication of individuals (typically, small modifications are more likely than large ones), while recombination exchanges information between two or more existing individuals. C3.2 C3.3

(iii) By means of evaluating individuals in their environment, a measure of quality or fitness value can be assigned to individuals. As a minimum requirement, a comparison of individual fitness is possible, yielding a binary decision (better or worse). According to the fitness measure, the *selection* process favors better individuals to reproduce more often than those that are relatively worse. C2

These are just the most general properties of evolutionary algorithms, and the instances of evolutionary algorithms as described in the following sections of this chapter use the components in various different ways and combinations. Some basic differences in the utilization of these principles characterize the mainstream instances of evolutionary algorithms; that is, *genetic algorithms*, *evolution strategies*, and *evolutionary programming*. See D B Fogel (1995) and Bäck (1996) for a detailed overview of similarities and differences of these instances and Bäck and Schwefel (1993) for a brief comparison. B1.2 B1.3 B1.4

- Genetic algorithms (originally described by Holland (1962, 1975) at Ann Arbor, Michigan, as so-called adaptive or reproductive plans) emphasize *recombination (crossover)* as the most important search operator and apply *mutation* with very small probability solely as a 'background operator.' They also use a probabilistic selection operator *(proportional selection)* and often rely on a *binary representation* of individuals. C3.3 C3.2 C2.2, C1.2

- Evolution strategies (developed by Rechenberg (1965, 1973) and Schwefel (1965, 1977) at the Technical University of Berlin) use normally distributed mutations to modify *real-valued vectors* and emphasize *mutation* and *recombination* as essential operators for searching in the search space and in the strategy parameter space at the same time. The *selection operator* is deterministic, and parent and offspring population sizes usually differ from each other. C1.3 C3.2.2, C3.3.2 C2.4
- Evolutionary programming (originally developed by Lawrence J Fogel (1962) at the University of California in San Diego, as described in Fogel *et al* (1966) and refined by David B Fogel (1992) and others) emphasizes mutation and does not incorporate the recombination of individuals. Similarly to evolution strategies, when approaching real-valued optimization problems, evolutionary programming also works with normally distributed mutations and extends the evolutionary process to the strategy parameters. The *selection operator* is probabilistic, and presently most applications are reported for search spaces involving real-valued vectors, but the algorithm was originally developed to evolve *finite-state machines*. C2.6.1 C1.5

In addition to these three mainstream methods, which are described in detail in the following sections, *genetic programming*, *classifier systems*, and *hybridizations* of evolutionary algorithms with other techniques are considered in this chapter. As an introductory remark, we only mention that genetic programming applies the evolutionary search principle to automatically develop computer programs in suitable *languages* (often LISP, but others are possible as well), while classifier systems search the space of production rules (or sets of rules) of the form '`IF <condition> THEN <action>`'. B1.5.1, B1.5.2, B1.5.3 C1.6

A variety of different representations of individuals and corresponding operators are presently known in evolutionary algorithm research, and it is the aim of Part C (Evolutionary Computation Models) to present all these in detail. Here, we will use Part C as a construction kit to assemble the basic instances of evolutionary algorithms.

As a general framework for these basic instances, we define I to denote an arbitrary space of individuals $\boldsymbol{a} \in I$, and $F : I \to \mathbb{R}$ to denote a real-valued fitness function of individuals. Using μ and λ to denote parent and offspring population sizes, $P(t) = (\boldsymbol{a}_1(t), \ldots, \boldsymbol{a}_\mu(t)) \in I^\mu$ characterizes a population at generation t. Selection, mutation, and recombination are described as operators $s : I^\lambda \to I^\mu$, $m : I^\kappa \to I^\lambda$, and $r : I^\mu \to I^\kappa$ that transform complete populations. By describing all operators on the population level (though this is counterintuitive for mutation), a high-level perspective is adopted, which is sufficently general to cover different instances of evolutionary algorithms. For mutation, the operator can of course be reduced to the level of single individuals by defining m through a multiple application of a suitable operator $m' : I \to I$ on individuals.

These operators typically depend on additional sets of parameters Θ_s, Θ_m, and Θ_r which are characteristic for the operator and the representation of individuals. Additionally, an initialization procedure generates a population of individuals (typically at random, but an initialization with known starting points should of course also be possible), an evaluation routine determines the fitness values of the individuals of a population, and a termination criterion is applied to determine whether or not the algorithm should stop.

Putting all this together, a basic evolutionary algorithm reduces to the simple recombination–mutation–selection loop as outlined below:

Input: $\mu, \lambda, \Theta_\iota, \Theta_r, \Theta_m, \Theta_s$
Output: $\boldsymbol{a}^*$, the best individual found during the run, or
P^*, the best population found during the run.

```
1   t ← 0;
2   P(t) ← initialize(μ);
3   F(t) ← evaluate(P(t), μ);
4   while (ι(P(t), Θ_ι) ≠ true) do
5       P'(t)   ← recombine(P(t), Θ_r);
6       P''(t)  ← mutate(P'(t), Θ_m);
7       F(t)    ← evaluate(P''(t), λ);
8       P(t+1)  ← select(P''(t), F(t), μ, Θ_s);
9       t       ← t + 1;
    od
```

After initialization of t (line 1) and the population $P(t)$ of size μ (line 2) as well as its fitness evaluation (line 3), the while-loop is entered. The termination criterion ι might depend on a variety of parameters, which are summarized here by the argument Θ_ι. Similarly, recombination (line 5), mutation

(line 6), and selection (line 8) depend on a number of algorithm-specific additional parameters. While $P(t)$ consists of μ individuals, $P'(t)$ and $P''(t)$ are assumed to be of size κ and λ, respectively. Of course, $\lambda = \kappa = \mu$ is allowed and is the default case in genetic algorithms. The setting $\kappa = \mu$ is also often used in evolutionary programming (without recombination), but it depends on the application and the situation is quickly changing. Either recombination or mutation might be absent from the main loop, such that $\kappa = \mu$ (absence of recombination) or $\kappa = \lambda$ (absence of mutation) is required in these cases. The selection operator selects μ individuals from $P''(t)$ according to the fitness values $\boldsymbol{F}(t)$, t is incremented (line 9), and the body of the main loop is repeated.

The input parameters of this general evolutionary algorithm include the population sizes μ and λ as well as the parameter sets Θ_t, Θ_r, Θ_m, and Θ_s of the basic operators. Notice that we allow recombination to equal the identity mapping; that is, $P''(t) = P'(t)$ is possible.

The following sections of this chapter present the common evolutionary algorithms as particular instances of the general scheme.

References

Bäck T 1996 *Evolutionary Algorithms in Theory and Practice* (New York: Oxford University Press)

Bäck T and Schwefel H-P 1993 An overview of evolutionary algorithms for parameter optimization *Evolutionary Computation* **1**(1) 1–23

Fogel D B 1992 *Evolving Artificial Intelligence* PhD Thesis, University of California, San Diego

——1995 *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence* (Piscataway, NJ: IEEE)

Fogel L J 1962 Autonomous automata *Industr. Res.* **4** 14–9

Fogel L J, Owens A J and Walsh M J 1966 *Artificial Intelligence through Simulated Evolution* (New York: Wiley)

Holland J H 1962 Outline for a logical theory of adaptive systems *J. ACM* **3** 297–314

——1975 *Adaptation in Natural and Artificial Systems* (Ann Arbor, MI: University of Michigan Press)

Rechenberg I 1965 Cybernetic solution path of an experimental problem *Library Translation No 1122* Royal Aircraft Establishment, Farnborough, UK

——1973 *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution* (Stuttgart: Frommann-Holzboog)

Schwefel H-P 1965 *Kybernetische Evolution als Strategie der experimentellen Forschung in der Strömungstechnik* Diplomarbeit, Technische Universität, Berlin

——1977 *Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie* Interdisciplinary Systems Research, vol 26 (Basel: Birkhäuser)

Further reading

The introductory section to evolutionary algorithms certainly provides the right place to mention the most important books on evolutionary computation and its subdisciplines. The following list is not intended to be complete, but only to guide the reader to the literature.

1. Bäck T 1996 *Evolutionary Algorithms in Theory and Practice* (New York: Oxford University Press)

   A presentation and comparison of evolution strategies, evolutionary programming, and genetic algorithms with respect to their behavior as parameter optimization methods. Furthermore, the role of mutation and selection in genetic algorithms is discussed in detail, arguing that mutation is much more useful than usually claimed in connection with genetic algorithms.

2. Goldberg D E 1989 *Genetic Algorithms in Search, Optimization, and Machine Learning* (Reading, MA: Addison-Wesley)

   An overview of genetic algorithms and classifier systems, discussing all important techniques and operators used in these subfields of evolutionary computation.

3. Rechenberg I 1994 *Evolutionsstrategie '94* Werkstatt Bionik und Evolutionstechnik, vol 1 (Stuttgart: Frommann-Holzboog)

   A description of evolution strategies in the form used by Rechenberg's group in Berlin, including a reprint of (Rechenberg 1973).

4. Schwefel H-P 1995 *Evolution and Optimum Seeking* Sixth-Generation Computer Technology Series (New York: Wiley)

   The most recent book on evolution strategies, covering the (μ, λ)-strategy and all aspects of self-adaptation of strategy parameters as well as a comparison of evolution strategies with classical optimization methods.

5. Fogel D B 1995 *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence* (Piscataway, NJ: IEEE)

   The book covers all three main areas of evolutionary computation (i.e. genetic algorithms, evolution strategies, and evolutionary programming) and discusses the potential for using simulated evolution to achieve machine intelligence.

6. Michalewicz Z 1994 *Genetic Algorithms + Data Structures = Evolution Programs* (Berlin: Springer)

   Michalewicz also takes a more general view at evolutionary computation, thinking of evolutionary heuristics as a principal method for search and optimization, which can be applied to any kind of data structure.

7. Kinnear K E 1994 *Advances in Genetic Programming* (Cambridge, MA: MIT Press)

   This collection of articles summarizes the state of the art in genetic programming, emphasizing other than LISP-based approaches to genetic programming.

8. Koza J R 1992 *Genetic Programming: On the Programming of Computers by Means of Natural Selection* (Cambridge, MA: MIT Press)

9. Koza J R 1994 *Genetic Programming II* (Cambridge, MA: MIT Press)

   The basic books for genetic programming using LISP programs, demonstrating the feasibility of the method by presenting a variety of application examples from diverse fields.

B1.2 Genetic algorithms

Larry J Eshelman

Abstract

This section gives an overview of genetic algorithms (GAs), describing the canonical GAs proposed by John Holland and developed by his first students, as well as later variations. Whereas many of the GA variations are distinguished by the methods used for selection, GAs as a class, including later variations, are distinguished from other evolutionary algorithms by their reliance upon crossover. Selection and crossover are discussed in some detail, and representation and parallelization are discussed briefly.

B1.2.1 Introduction

Genetic algorithms (GAs) are a class of evolutionary algorithms first proposed and analyzed by John Holland (1975). There are three features which distinguish GAs, as first proposed by Holland, from other evolutionary algorithms: (i) the representation used—*bitstrings*; (ii) the method of selection—*proportional selection*; and (iii) the primary method of producing variations—*crossover*. Of these three features, however, it is the emphasis placed on crossover which makes GAs distinctive. Many subsequent GA implementations have adopted alternative methods of selection, and many have abandoned bitstring representations for other representations more amenable to the problems being tackled. Although many alternative methods of crossover have been proposed, in almost every case these variants are inspired by the spirit which underlies Holland's original analysis of GA behavior in terms of the processing of schemata or building blocks. It should be pointed out, however, that the *evolution strategy* paradigm has added crossover to its repertoire, so that the distinction between classes of evolutionary algorithms has become blurred (Bäck *et al* 1991).

C1.2 C2.2, C3.3 B1.3

We shall begin by outlining what might be called the canonical GA, similar to that described and analyzed by Holland (1975) and Goldberg (1987). We shall introduce a framework for describing GAs which is richer than needed but which is convenient for describing some variations with regard to the method of selection. First we shall introduce some terminology. The individual structures are often referred to as chromosomes. They are the genotypes that are manipulated by the GA. The evaluation routine decodes these structures into some phenotypical structure and assigns a fitness value. Typically, but not necessarily, the chromosomes are bitstrings. The value at each locus on the bitstring is referred to as an allele. Sometimes the individuals loci are also called genes. At other times genes are combinations of alleles that have some phenotypical meaning, such as parameters.

B1.2.2 Genetic algorithm basics and some variations

An initial population of individual structures $P(0)$ is generated (usually randomly) and each individual is evaluated for fitness. Then some of these individuals are selected for mating and copied (select_repro) to the mating buffer $C(t)$. In Holland's original GA, individuals are chosen for mating probabilistically, assigning each individual a probability proportional to its observed performance. Thus, better individuals are given more opportunities to produce offspring (reproduction with emphasis). Next the genetic operators (usually mutation and crossover) are applied to the individuals in the mating buffer, producing offspring $C'(t)$. The rates at which mutation and crossover are applied are an implementation decision.

If the rates are low enough, it is likely that some of the offspring produced will be identical to their parents. Other implementation details are how many offspring are produced by crossover (one or two), and how many individuals are selected and paired in the mating buffer. In Holland's original description, only one pair is selected for mating per cycle. The pseudocode for the genetic algorithm is as follows:

```
begin
    t = 0;
    initialize P(t);
    evaluate structures in P(t);
    while termination condition not satisfied do
    begin
        t = t + 1;
        select_repro C(t) from P(t-1);
        recombine and mutate structures in C(t) forming C'(t);
        evaluate structures in C'(t);
        select_replace P(t) from C'(t) and P(t-1);
    end
end
```

After the new offspring have been created via the genetic operators the two populations of parents and children must be merged to create a new population. Since most GAs maintain a fixed-sized population M, this means that a total of M individuals need to be selected from the parent and child populations to create a new population. One possibility is to use all the children generated (assuming that the number is not greater than M) and randomly select (without any bias) individuals from the old population to bring the new population up to size M. If only one or two new offspring are produced, this in effect means randomly replacing one or two individuals in the old population with the new offspring. (This is what Holland's original proposal did.) On the other hand, if the number of offspring created is equal to M, then the old parent population is completely replaced by the new population.

There are several opportunities for biasing selection: selection for reproduction (or mating) and selection from the parent and child populations to produce the new population. The GAs most closely associated with Holland do all their biasing at the reproduction selection stage. Even among these GAs, however, there are a number of variations. If reproduction with emphasis is used, then the probability of an individual being chosen is a function of its observed fitness. A straightforward way of doing this would be to total the fitness values assigned to all the individuals in the parent population and calculate the probability of any individual being selected by dividing its fitness by the total fitness. One of the properties of this way of assigning probabilities is that the GA will behave differently on functions that seem to be equivalent from an optimization point of view such as $y = ax^2$ and $y = ax^2 + b$. If the b value is large in comparison to the differences in the value produced by the ax^2 term, then the differences in the probabilities for selecting the various individuals in the population will be small, and selection pressure will be very weak. This often happens as the population converges upon a narrow range of values. One way of avoiding this behavior is to *scale* the fitness function, typically to the worst individual in the population (De Jong 1975). Hence the measure of fitness used in calculating the probability for selecting an individual is not the individual's absolute fitness, but its fitness relative to the worst individual in the population. C2.2.2

Although scaling can eliminate the problem of not enough selection pressure, often GAs using fitness proportional selection suffer from the opposite problem—too much selection pressure. If an individual is found which is much better than any other, the probability of selecting this individual may become quite high (especially if scaling to the worst is used). There is the danger that many copies of this individual will be placed in the mating buffer, and this individual (and its similar offspring) will rapidly take over the population (premature convergence). One way around this is to replace fitness proportional selection with *ranked selection* (Whitley 1989). The individuals in the parent population are ranked, and the probability of selection is a linear function of rank rather than fitness, where the 'steepness' of this function is an adjustable parameter. C2.4

Another popular method of performing selection is *tournament selection* (Goldberg and Deb 1991). A small subset of individuals is chosen at random, and then the best individual (or two) in this set is (are) selected for the mating buffer. Tournament selection, like rank selection, is less subject to rapid takeover by good individuals, and the selection pressure can be adjusted by controlling the size of the subset used. C2.3

Another common variation of those GAs that rely upon reproduction selection for their main source of selection bias is to maintain one copy of the best individual found so far (De Jong 1975). This is referred to as the *elitist strategy*. It is actually a method of biased parent selection, where the best member of the parent population is chosen and all but one of the M members of the child population are chosen. Depending upon the implementation, the selection of the child to be replaced by the best individual from the parent population may or may not be biased. C2.7.4

A number of GA variations make use of biased replacement selection. Whitley's GENITOR, for example, creates one child each cycle, selecting the parents using ranked selection, and then replacing the worst member of the population with the new child (Whitley 1989). Syswerda's steady-state GA creates two children each cycle, selecting parents using ranked selection, and then stochastically choosing two individuals to be replaced, with a bias towards the worst individuals in the parent population (Syswerda 1989). Eshelman's CHC uses unbiased reproductive selection by randomly pairing all the members of the parent population, and then replacing the worst individuals of the parent population with the better individuals of the child population. (In effect, the offspring and parent populations are merged and the best M (population size) individuals are chosen.) Since the new offspring are only chosen by CHC if they are better than the members of the parent population, the selection of both the offspring and parent populations is biased (Eshelman 1991).

These methods of replacement selection, and especially that of CHC, resemble the $(\mu+\lambda)$ ES method of selection sometimes originally used by evolution strategies (ESs) (Bäck *et al* 1991). From μ parents λ offspring are produced; the μ parents and λ offspring are merged; and the best μ individuals are chosen to form the new parent population. The other ES selection method, (μ, λ) ES, places all the bias in the child selection stage. In this case, μ parents produce λ offspring ($\lambda > \mu$), and the best μ offspring are chosen to replace the parent population. Mühlenbein's breeder GA also uses this selection mechanism (Mühlenbein and Schlierkamp-Voosen 1993). C2.4.4 C2.4.4

Often a distinction is made between *generational* and *steady-state* GAs. Unfortunately, this distinction tends to merge two properties that are quite independent: whether the replacement strategy of the GA is biased or not and whether the GA produces one (or two) versus many (usually M) offspring each cycle. Syswerda's steady-state GA, like Whitley's GENITOR, allows only one mating per cycle and uses a biased replacement selection, but there are also GAs that combine multiple matings per cycle with biased replacement selection (CHC) as well as a whole class of ESs ($(\mu+\lambda)$ ES). Furthermore, the GA described by Holland (1975) combined a single mating per cycle and unbiased replacement selection. Of these two features, it would seem that the most significant is the replacement strategy. De Jong and Sarma (1993) found that the main difference between GAs allowing many matings versus few matings per cycle is that the latter have a higher variance in performance. C2.7.3

The choice between a biased and an unbiased replacement strategy, on the other hand, is a major determinant of GA behavior. First, if biased replacement is used in combination with biased reproduction, then the problem of premature convergence is likely to be compounded. (Of course this will depend upon other factors, such as the size of the population, whether ranked selection is used, and, if so, the setting of the selection bias parameter.) Second, the obvious shortcoming of unbiased replacement selection can turn out to be a strength. On the negative side, replacing the parents by the children, with no mechanism for keeping those parents that are better than any of the children, risks losing, perhaps forever, very good individuals. On the other hand, replacing the parents by the children can allow the algorithm to wander, and it may be able to wander out of a local minimum that would trap a GA relying upon biased replacement selection. Which is the better strategy cannot be answered except in the context of the other mechanisms of the algorithm (as well as the nature of the problem being solved). Both Syswerda's steady-state GA and Whitley's GENITOR combine a biased replacement strategy with a mechanism for eliminating children which are duplicates of any member in the parent population. CHC uses unbiased reproductive selection, relying solely upon biased replacement selection as its only source of selection pressure, and uses several mechanisms for maintaining diversity (not mating similar individuals and seeded restarts), which allow it to take advantage of the preserving properties of a deterministic replacement strategy without suffering too severely from its shortcomings.

B1.2.3 Mutation and crossover

All evolutionary algorithms work by combining selection with a mechanism for producing variations. The best known mechanism for producing variations is *mutation*, where one allele of a gene is randomly C3.2

replaced by another. In other words, new trial solutions are created by making small, random changes in the representation of prior trial solutions. If a binary representation is used, then mutation is achieved by 'flipping' bits at random. A commonly used rate of mutation is one over the string length. For example, if the chromosome is one hundred bits long, then the mutation rate is set so that each bit has a probability of 0.01 of being flipped.

Although most GAs use mutation along with crossover, mutation is sometimes treated as if it were a background operator for assuring that the population will consist of a diverse pool of alleles that can be exploited by crossover. For many optimization problems, however, an evolutionary algorithm using mutation without crossover can be very effective (Mathias and Whitley 1994). This is not to suggest that crossover never provides an added benefit, but only that one should not disparage mutation.

The intuitive idea behind crossover is easy to state: given two individuals who are highly fit, but for different reasons, ideally what we would like to do is create a new individual that combines the best features from each. Of course, since we presumably do not know which features account for the good performance (if we did we would not need a search algorithm), the best we can do is to recombine features at random. This is how crossover operates. It treats these features as building blocks scattered throughout the population and tries to recombine them into better individuals via crossover. Sometimes crossover will combine the worst features from the two parents, in which case these children will not survive for long. But sometimes it will recombine the best features from two good individuals, creating even better individuals, provided these features are compatible.

Suppose that the representation is the classical bitstring representation: individual solutions in our population are represented by binary strings of zeros and ones of length L. A GA creates new individuals via crossover by choosing two strings from the parent population, lining them up, and then creating two new individuals by swapping the bits at random between the strings. (In some GAs only one individual is created and evaluated, but the procedure is essentially the same.) Holland originally proposed that the swapping be done in segments, not bit by bit. In particular, he proposed that a single locus be chosen at random and all bits after that point be swapped. This is known as *one-point crossover*. Another C3.3 common form of crossover is two-point crossover which involves choosing two points at random and swapping the corresponding segments from the two parents defined by the two points. There are of course many possible variants. The best known alternative to one- and two-point crossover is *uniform crossover*. Uniform crossover randomly swaps individual bits between the two parents (i.e. exchanges between the parents the values at loci chosen at random).

Following Holland, GA behavior is typically analyzed in terms of *schemata*. Given a space of B2.5 structures represented by bitstrings of length L, schemata represent partitions of the search space. If the bitstrings of length L are interpreted as vectors in a L-dimensional hypercube, then schemata are hyperplanes of the space. A schema can be represented by a string of L symbols from the set $0, 1, \#$ where # is a 'wildcard' matching either 0 or 1. Each string of length L may be considered a sample from the partition defined by a schema if it matches the schema at each of the defined positions (i.e. the non-# loci). For example, the string 011001 instantiates the schema 01##0#. Each string, in fact, instantiates 2^L schemata.

Two important schema properties are order and defining length. The order of a schema is the number of defined loci (i.e. the number of non-# symbols). For example the schema #01##1### is an order 3 schema. The defining length is the distance between the loci of the first and last defined positions. The defining length of the above schema is four since the loci of the first and last defined positions are 2 and 6.

From the hyperplane analysis point of view, a GA can be interpreted as focusing its search via crossover upon those hyperplane partition elements that have on average produced the best-performing individuals. Over time the search becomes more and more focused as the population converges since the degree of variation used to produce new offspring is constrained by the remaining variation in the population. This is because crossover has the property that Radcliffe refers to as respect—if two parents are instances of the same schema, the child will also be an instance (Radcliffe 1991). If a particular schema conveys high fitness values to its instances, then the population is likely to converge on the defining bits of this schema. Once it so converges, all offspring will be instances of this schema. This means that as the population converges, the search becomes more and more focused on smaller and smaller partitions of the search space.

It is useful to contrast crossover with mutation in this regard. Whereas mutation creates variations by flipping bits randomly, crossover is restricted to producing variations at those loci on which the population

has not yet converged. Thus crossover, and especially bitwise versions of crossover, can be viewed as a form of adaptive mutation, or convergence-controlled variation (CCV).

The standard explanation of how GAs operate is often referred to as the *building block hypothesis*. B2.5.3 According to this hypothesis, GAs operate by combining small building blocks into larger building blocks. The intuitive idea behind recombination is that by combining features (or building blocks) from two good parents crossover will often produce even better children; for example, a mother with genes for sharp teeth and a father with genes for sharp claws will have the potential of producing some children who have both features. More formally, the building blocks are the schemata discussed above.

Loosely interpreted, the building block hypothesis is another way of asserting that GAs operate through a process of CCV. The building block hypothesis, however, is often given a stronger interpretation. In particular, crossover is seen as having the added value of being able to recombine middle-level building blocks that themselves cannot be built from lower-level building blocks (where *level* refers to either the defining length or order, depending on the crossover operator). We shall refer to this explanation as to how GAs work as the strict building block hypothesis (SBBH), and contrast it with the weaker convergence-controlled variation hypothesis (CCVH).

To differentiate these explanations, it is useful to compare crossover with an alternative mechanism for achieving CCV. Instead of pairing individuals and swapping segments or bits, a more direct method of generating CCVs is to use the distribution of the allele values in the population to generate new offspring. This is what Syswerda's bitwise simulated crossover (BSC) algorithm does (Syswerda 1993). In effect, the distribution of allele values is used to generate a vector of allele probabilities, which in turn is used to generate a string of ones and zeros. Baluja's PBIL goes one step further and eliminates the population, and simply keeps a probability vector of allele values, using an update rule to modify it based on the fitness of the samples generated (Baluja 1995).

The question is, if one wants to take advantage of CCV with its ability to adapt, why use crossover, understood as involving pairwise mating, rather than one of these *poolwise* schemes? One possible answer is that the advantage is only one of implementation. The pairwise implementation does not require any centralized bookkeeping mechanism. In other words, crossover (using pairwise mating) is simply nature's way of implementing a decentralized version of CCV.

A more theoretically satisfying answer is that pairwise mating is better able to preserve essential linkages among the alleles. One manifestation of this is that there is no obvious way to implement a segment-based version of poolwise mating, but this point also applies if we compare poolwise mating with only crossover operators that operate at the bit level, such as uniform crossover. If two allele values are associated in some individual, the probability of these values being associated in the children is much higher for pairwise mating than poolwise. To see this consider an example. Suppose the population size is 100, and that an individual of average fitness has some unique combination of allele values, say all ones in the first three positions. This individual will have a 0.01 probability (one out of 100) of being selected for mating, assuming it is of average fitness. If uniform crossover is being used, with a 0.5 probability of swapping the values at each locus, and one offspring is being produced per mating, then the probability of the three allele values being propagated without disruption has a lower bound of 0.125 (0.5^3). This is assuming the worst-case scenario that every other member in the population has all zeros in the first three positions (and ignoring the possibility of mating this individual with a copy of itself). Thus, the probability of propagating this schema is 0.00125 ($0.01 * 0.125$). On the other hand, if BSC is being used, then the probability of propagating this schema is much lower. Since there is only one instance of this individual in the population, there is only one chance in 100 of propagating each allele and only 0.000 001 (0.01^3) of propagating all three.

Ultimately, one is faced with a tradeoff: the enhanced capability of pairwise mating to propagate difficult-to-find schemata is purchased at the risk of increased hitchhiking; that is, the population may prematurely converge on bits that do not convey additional fitness but happen to be present in the individuals that are instances of good schemata. According to both the CCVH and the SBBH, crossover must not simply preserve and propagate good schemata, but must also recombine them with other good schemata. Recombination, however, requires that these good schemata be tried in the context of other schemata. In order to determine which schemata are the ones contributing to fitness, we must test them in many different contexts, and this involves prying apart the defining positions that contribute to fitness from those that are spurious, but the price for this reduced hitchhiking is higher disruption (the breaking up of the good schemata). This price will be too high if the algorithm cannot propagate critical, highly valued, building blocks or, worse yet, destroys them in the next crossover cycle.

This tradeoff applies not only to the choice between poolwise and pairwise methods of producing variation, but also to the choice between various methods of crossover. Uniform crossover, for example, is less prone to hitchhiking than two-point crossover, but is also more disruptive, and poolwise mating schemes are even more disruptive than uniform crossover. In Holland's original analysis this tradeoff between preserving the good schemata while performing vigorous recombination is downplayed by using a segment-based crossover operator such as one- or two-point crossover and assuming that the important building blocks are of short defining length. Unfortunately, for the types of problem to which GAs are supposedly ideally suited—those that are highly complex with no tractable analytical solution—there is no *a priori* reason to assume that the problem will, or even can, be represented so that important building blocks will be those with short defining length. To handle this problem Holland proposed an inversion operator that could reorder the loci on the string, and thus be capable of finding a representation that had building blocks with short defining lengths. The inversion operator, however, has not proven sufficiently effective in practice at recoding strings on the fly. To overcome this linkage problem, Goldberg has proposed what he calls *messy GAs*, but, before discussing messy GAs, it will be helpful to describe a class C4.2.4 of problems that illustrate these linkage issues: deceptive problems.

Deception is a notion introduced by Goldberg (1987). Consider two incompatible schemata, A and B2.7.1 B. A problem is deceptive if the average fitness of A is greater than B even though B includes a string that has a greater fitness than any member of A. In practice this means that the lower-order building blocks lead the GA away from the global optimum. For example, consider a problem consisting of five-bit segments for which the fitness of each is determined as follows (Liepins and Vose 1991). For each *one* the segment receives a point, and thus five points for all *ones*, but for all *zeros* it receives a value greater than five. For problems where the value of the optimum is between five and eight the problem is fully deceptive (i.e. all relevant lower-order hyperplanes lead toward the deceptive attractor). The total fitness is the sum of the fitness of the segments.

It should be noted that it is probably a mistake to place too much emphasis on the formal definition of deception (Grefenstette 1993). What is really important is the concept of being misled by the lower-order building blocks. Whereas the formal definition of deception stresses the average fitness of the hyperplanes taken over the entire search space, selection only takes into account the observed average fitness of hyperplanes (those in the actual population). The interesting set of problems is those that are misleading in that manipulation of the lower-order building blocks is likely to lead the search away from the middle-level building blocks that constitute the optimum solution, whether these middle-level building blocks are deceptive in the formal sense or not. In the above class of functions, even when the value of the optimum is greater than eight (and so not fully deceptive), but still not very large, e.g. ten, the problem is solvable by a GA using segment-based crossover, very difficult for a GA using bitwise uniform crossover, and all but impossible for a poolwise-based algorithm like BSC.

As long as the deceptive problem is represented so that the loci of the positions defining the building blocks are close together on the string, it meets Holland's original assumption that the important building blocks are of short defining length. The GA will be able to exploit this information using one- or two-point crossover—the building blocks will have a low probability of being disrupted, but will be vigorously recombined with other building blocks along the string. If, on the other hand, the bits constituting the deceptive building blocks are maximally spread out on the chromosome, then a crossover operator such as one- or two-point crossover will tend to break up the good building blocks. Of course, maximally spreading the deceptive bits along the string is the extreme case, but bunching them together is the opposite extreme.

Since one is not likely to know enough about the problem to be able to guarantee that the building blocks are of short defining length, segmented crossover loses its advantage over bitwise crossover. It is true that bitwise crossover operators are more disruptive, but there are several solutions to this problem. First, there are bitwise crossover operators that are much less disruptive than the standard uniform crossover operator (Spears and De Jong 1991, Eshelman and Schaffer 1995). Second, the problem of preservation can often be ameliorated by using some form of replacement selection so that good individuals survive until they are replaced by better individuals (Eshelman and Schaffer 1995). Thus a disruptive form of crossover such as uniform crossover can be used and good schemata can still be preserved. Uniform crossover will still make it difficult to propagate these high-order, good schemata once they are found, but, provided the individuals representing these schemata are not replaced by better individuals that represent incompatible schemata, they will be preserved and may eventually be able to propagate their schemata on to their offspring. Unfortunately, this proviso is not likely to be met by any but low-order deceptive problems. Even for deceptive problems of order five, the difficulty of propagating optimal schemata is

such that the suboptimal schemata tend to crowd out the optimum ones.

Perhaps the ultimate GA for tackling deceptive problems is Goldberg's messy GA (mGA) (Goldberg *et al* 1991). Whereas in more traditional GAs the manipulation of building blocks is implicit, mGAs explicitly manipulate the building blocks. This is accomplished by using variable-length strings that may be underspecified or overspecified; that is, some bit positions may not be defined, and some positions may have conflicting specifications. This is what makes mGAs messy.

These strings constitute the building blocks. They consist of a set of position–value pairs. Overspecified strings are evaluated by a simple conflict resolution strategy such as first-come-first-served rules. Thus, ((1 0) (2 1) (1 1) (3 0)) would be interpreted as 010, ignoring the third pair, since the first position has already been defined. Underspecified strings are interpreted by filling in the missing values using a competitive template, a locally optimal structure. For example, if the locally optimal structure, found by testing one bit at a time, is 111, then the string ((1 0) (3 0)) would be interpreted by filling in the value for the (missing) second position with the value of the second position in the template. The resulting 010 string would then be evaluated.

mGAs have an outer and an inner loop. The inner loop consists of three phases: the initialization, primordial, and juxtaposition phases. In the initialization phase all substrings of length k are created and evaluated, i.e. all combinations of strings with k defining positions (where k is an estimate of the highest order of deception in the problem). As was explained above the missing values are filled in using the competitive template. (As will be explained below, the template for the k level of the outer loop is the solution found at the $k-1$ level.)

In the primordial phase, selection is applied to the population of individuals produced during the initialization phase without any operators. Thus the substrings that have poor evaluations are eliminated and those with good evaluations have multiple copies in the resulting population.

In the juxtapositional phase selection in conjunction with cut and splice operators is used to evolve improved variations. Again, the competitive template is used for filling in missing values, and the first-come-first-served rule is used for handling overspecified strings created by the splice operator. The cut and splice operators act much like one-point crossover in a traditional GA, keeping in mind that the strings are of variable length and may be underspecified or overspecified.

The outer loop is over levels. It starts at the level of $k = 1$, and continues through each level until it reaches a user-specified stopping criterion. At each level, the solution found at the previous level is used as the competitive template.

One of the limitations of mGAs as originally conceived is that the initialization phase becomes extremely expensive as the mGA progresses up the levels. A new variant of the mGA speeds up the process by eliminating the need to process all the variants in the initialization stage (Goldberg *et al* 1993). The initialization and primordial phases of the original mGA are replaced by a 'probabilistically complete initialization' procedure. This procedure is divided into several steps. During the first step strings of nearly length L are evaluated (using the template to fill in the missing values). Then selection is applied to these strings without any operators (much as was done in the primordial phase of the original mGA, but for only a few generations). Then the algorithm enters a filtering step where some of the genes in the strings are deleted, and the shortened strings are evaluated using the competitive template. Then selection is applied again. This process is repeated until the resulting strings are of length k. Then the mGA goes into the juxtaposition stage like the original mGA. By replacing the original initialization and primordial stages with stepwise filtering and selection, the number of evaluations required is drastically reduced for problems of significant size. (Goldberg *et al* (1993) provide analytical methods for determining the population and filtering reduction constants.) This new version of the mGA is very effective at solving loosely linked deceptive problems, i.e. those problems where the defining positions of the deceptive segments are spread out along the bitstring.

mGAs were designed to operate according to the SBBH, and deceptive problems illustrate that there are problems where being able to manipulate building blocks can provide an added value over CCV. It still is an open question, however, as to how representative deceptive problems are of the types of real-world problem that GAs might encounter. No doubt, many difficult real-world problems have deceptive or misleading elements in them. If they did not, they could be easily solved by local search methods. However it does not necessarily follow that such problems can be solved by a GA that is good at solving deceptive problems. The SBBH assumes that the misleading building blocks will exist in the initial population, that they can be identified early in the search before they are lost, and that the problem can be solved incrementally by combining these building blocks, but perhaps the building blocks that have

misleading alternatives have little meaning until late in the search and so cannot be expected to survive in the population.

Even if the SBBH turns out not to be as useful an hypothesis as originally supposed, the increased propagation capabilities of pairwise mating may give a GA (using pairwise mating) an advantage over a poolwise CCV algorithm. To see why this is the case it is useful to define the prototypical individual for a given population: for each locus we assign a one or a zero depending upon which value is most frequent in the population (randomly assigning a value if they are equally frequent). Suppose the population contains some maverick individual that is quite far from the prototypical individual although it is near the optimum (as measured by Hamming distance) but is of only average fitness. Since an algorithm using a poolwise method of producing offspring will tend to produce individuals that are near the prototypical individual, such an algorithm is unlikely to explore the region around the maverick individual. On the other hand, a GA using pairwise mating is more likely to explore the region around the maverick individual, and so more likely to discover the optimum. Ironically, pairwise mating is, in this respect, more mutation-like than poolwise mating. While pairwise mating retains the benefits of CCV, it less subject to the majoritarian tendencies of poolwise mating.

B1.2.4 Representation

Although GAs typically use a bitstring representation, GAs are not restricted to bitstrings. A number of early proponents of GAs developed GAs that use other representations, such as *real-valued parameters* C1.3 (Davis 1991, Janikow and Michalewicz 1991, Wright 1991), *permutations* C1.4 (Davis 1985, Goldberg and Lingle 1985, Grefenstette *et al* 1985), and *treelike hierarchies* C1.6 (Antonisse and Keller 1987). Koza's *genetic programming* (GP) paradigm (Koza 1992) is a GA-based method for evolving programs, where B1.5.1 the data structures are LISP S-expressions, and crossover creates new LISP S-expressions (offspring) by exchanging subtrees from the two parents.

In the case of combinatorial problems such as the *traveling salesman problem* (TSP), a number of G9.5 order-based or sequencing crossover operators have been proposed. The choice of operator will depend upon one's goal. If the goal is to solve a TSP, then preserving adjacency information will be the priority, which suggests a crossover operator that operates on common edges (links between cities shared by the two parents) (Whitley *et al* 1989). On the other hand, if the goal is to solve a *scheduling* problem, then F1.5 preserving relative order is likely to be the priority, which suggests an order preserving crossover operator. Syswerda's order crossover operator (Syswerda 1991), for example, chooses several positions at random in the first parent, and then produces a child so that the relative order of the chosen elements in the first parent is imposed upon the second parent.

Even if binary strings are used, there is still a choice to be made as to which binary coding scheme to use for numerical parameters. Empirical studies have usually found that Gray code is superior to the standard power-of-two binary coding (Caruana and Schaffer 1988), at least for the commonly used test problems. One reason is that the latter introduces Hamming cliffs—two numerically adjacent values may have bit representations that are many bits apart (up to $L - 1$). This will be a problem if there is some degree of gradualness in the function, i.e. small changes in the variables usually correspond to small changes in the function. This is often the case for functions with numeric parameters.

As an example, consider a five-bit parameter, with a range from 0 to 31. If it is encoded using the standard binary coding, then 15 is encoded as 01111, whereas 16 is encoded as 10000. In order to move from 15 to 16, all five bits need to be changed. On the other hand, using Gray coding, 15 would be represented as 01000 and 16 as 11000, differing only by 1 bit.

When choosing an alternative representation, it is critical that a crossover operator be chosen that is appropriate for the representation. For example, if real-valued parameters are used, then a possible crossover operator is one that for each parameter uses the parameter values of the two parents to define an interval from which a new parameter is chosen (Eshelman and Schaffer 1993). As the GA makes progress it will narrow the range over which it searches for new parameter values.

If, for the chosen representation and crossover operator, the building blocks are unlikely to be instantiated independently of each other in the population, then a GA may not be appropriate. This problem has plagued finding crossover operators that are good for solving TSPs. The natural building blocks, it would seem, are subtours. However, what counts as a good subtour will almost always depend upon what the other subtours are. In other words, two good, but suboptimal solutions to a TSP may not

have many subtours (other than very short ones) that are compatible with each other so that they can be spliced together to form a better solution. This hurdle is not unique to combinatorial problems.

Given the importance of the representation, a number of researches have suggested methods for allowing the GA to adapt its own coding. We noted earlier that Holland proposed the inversion operator for rearranging the loci in the string. Another approach to adapting the representation is Shaefer's ARGOT system (Shaefer 1987). ARGOT contains an explicit parameterized representation of the mappings from bitstrings to real numbers and heuristics for triggering increases and decreases in resolution and for shifts in the ranges of these mappings. A similar idea is employed by Schraudolph and Belew (1992) who provide a heuristic for increasing the resolution triggered when the population begins to converge. Mathias and Whitley (1994) have proposed what they call delta coding. When the population converges, the numeric representation is remapped so that the parameter ranges are centered around the best value found so far, and the algorithm is restarted. There are also heuristics for narrowing or extending the range.

There are also GAs with mechanisms for dynamically adapting the rate at which GA operators are used or which operator is used. Davis, who has developed a number of nontraditional operators, proposed a mechanism for adapting the rate at which these operators are applied based on the past success of these operators during a run of the algorithm (Davis 1987).

B1.2.5 Parallel genetic algorithms

All evolutionary algorithms, because they maintain a population of solutions, are naturally parallelizable. However, because GAs use crossover, which is a way of sharing information, there are two other variations that are unique to GAs (Gordon and Whitley 1993). The first, most straightforward, method is to simply have one global population with multiple processors for evaluating individual solutions. The second method, often referred to as the *island model* (alternatively, the migration or coarse-grain model), maintains C6.3 separate subpopulations. Selection and crossover take place in each subpopulation in isolation from the other subpopulations. Every so often an individual from one of the subpopulations is allowed to migrate to another subpopulation. This way information is shared among subpopulations.

The third method, often referred to as the *neighborhood model* (alternatively, the diffusion or fine-grain C6.4 model), maintains overlapping neighborhoods. The neighborhood for which selection (for reproduction and replacement) applies is restricted to a region local to each individual. What counts as a neighborhood will depend upon the neighborhood topology used. For example, if the population is arranged upon some type of spherical structure, individuals might be allowed to mate with (and forced to compete with) neighbors within a certain radius.

B1.2.6 Conclusion

Although the above discussion has been in the context of GAs as potential function optimizers, it should be pointed out that Holland's initial GA work was in the broader context of exploring GAs as adaptive systems (De Jong 1993). GAs were designed to be a simulation of evolution, not to solve problems. Of course, evolution has come up with some wonderful designs, but one must not lose sight of the fact that evolution is an opportunistic process operating in an environment that is continuously changing. Simon has described evolution as a process of searching where there is no goal (Simon 1983). This is not to question the usefulness of GAs as function optimizers, but only to emphasize that the perspective of function optimization is somewhat different from that of adaptation, and that the requirements of the corresponding algorithms will be somewhat different.

References

Antonisse H J and Keller K S 1987 Genetic operators for high-level knowledge representations *Proc. 2nd Int. Conf. on Genetic Algorithms (Cambridge, MA, 1987)* ed J J Grefenstette (Hillsdale, NJ: Erbaum) pp 69–76

Bäck T, Hoffmeister F and Schwefel H 1991 A survey of evolution strategies *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, 1991)* ed R K Belew and L B Booker (San Mateo, CA: Morgan Kaufmann) pp 2–9

Baluja S 1995 *An Empirical Comparison of Seven Iterative and Evolutionary Function Optimization Heuristics* Carnegie Mellon University School of Computer Science Technical Report CMU-CS-95-193

Caruana R A and Schaffer J D 1988 Representation and hidden bias: Gray vs. binary coding for genetic algorithms *Proc. 5th Int. Conf. on Machine Learning* (San Mateo, CA: Morgan Kaufmann) pp 153–61

Davis L 1985 Applying adaptive algorithms to epistatic domains *Proc. Int. Joint Conference on Artificial Intelligence* pp 162–4

——1987 Adaptive operator probabilities in genetic algorithms *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 61–9

——1991 Hybridization and numerical representation *The Handbook of Genetic Algorithms* ed L Davis (New York: Van Nostrand Reinhold) pp 61–71

De Jong K 1975 *An Analysis of the Behavior of a Class of Genetic Adaptive Systems* Doctoral Thesis, Department of Computer and Communication Sciences, University of Michigan

——1993 Genetic algorithms are not function optimizers *Foundations of Genetic Algorithms 2* ed D Whitley (San Mateo, CA: Morgan Kaufmann) pp 5–17

De Jong K and Sarma J 1993 Generation gaps revisited *Foundations of Genetic Algorithms 2* ed D Whitley (San Mateo, CA: Morgan Kaufmann) pp 19–28

Eshelman L J 1991 The CHC adaptive search algorithm: how to have safe search when engaging in nontraditional genetic recombination *Foundations of Genetic Algorithms* ed G J E Rawlins (San Mateo, CA: Morgan Kaufmann) pp 265–83

Eshelman L J and Schaffer J D 1993 Real-coded genetic algorithms and interval schemata *Foundations of Genetic Algorithms 2* ed D Whitley (San Mateo, CA: Morgan Kaufmann) pp 187–202

——1995 Productive recombination and propagating and preserving schemata *Foundations of Genetic Algorithms 3* ed D Whitley (San Mateo, CA: Morgan Kaufmann) pp 299–313

Goldberg D E 1987 Simple genetic algorithms and the minimal, deceptive problem *Genetic Algorithms and Simulated Annealing* ed L Davis (San Mateo, CA: Morgan Kaufmann) pp 74–88

——1989 *Genetic Algorithms in Search, Optimization, and Machine Learning* (Reading, MA: Addison-Wesley)

Goldberg D E and Deb K 1991 A comparative analysis of selection schemes used in genetic algorithms *Foundations of Genetic Algorithms* ed G J E Rawlins (San Mateo, CA: Morgan Kaufmann) pp 69–93

Goldberg D E, Deb K, Kargupta H and Harik G 1993 Rapid, accurate optimization of difficult problems using fast messy genetic algorithms *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 56–64

Goldberg D E, Deb K and Korb B 1991 Don't worry, be messy *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, 1991)* ed R K Belew and L B Booker (San Mateo, CA: Morgan Kaufmann) pp 24–30

Goldberg D E and Lingle R L 1985 Alleles, loci, and the traveling salesman problem *Proc. 1st Int. Conf. on Genetic Algorithms (Pittsburgh, PA, 1985)* ed J J Grefenstette (Hillsdale, NJ: Erbaum) pp 154–9

Gordon V S and Whitley 1993 Serial and parallel genetic algorithms and function optimizers *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 177–83

Grefenstette J J 1993 Deception considered harmful *Foundations of Genetic Algorithms 2* ed D Whitley (San Mateo, CA: Morgan Kaufmann) pp 75–91

Grefenstette J J, Gopal R, Rosmaita B J and Van Gucht D 1985 Genetic algorithms for the traveling salesman problem *Proc. 1st Int. Conf. on Genetic Algorithms (Pittsburgh, PA, 1985)* ed J J Grefenstette (Hillsdale, NJ: Erbaum) pp 160–8

Holland J H 1975 *Adaptation in Natural and Artificial Systems* (Ann Arbor, MI: University of Michigan Press)

Janikow C Z and Michalewicz Z 1991 An experimental comparison of binary and floating point representations in genetic algorithms *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, 1991)* ed R K Belew and L B Booker (San Mateo, CA: Morgan Kaufmann) pp 31–6

Koza J 1992 *Genetic Programming: on the Programming of Computers by Means of Natural Selection and Genetics* (Cambridge, MA: MIT Press)

Liepins G E and Vose M D 1991 Representational issues in genetic optimization *J. Exp. Theor. AI* **2** 101–15

Mathias K E and Whitley L D 1994 Changing representations during search: a comparative study of delta coding *Evolutionary Comput.* **2**

Mühlenbein H and Schlierkamp-Voosen 1993 The science of breeding and its application to the breeder genetic algorithm *Evolutionary Comput.* **1**

Radcliffe N J 1991 Forma analysis and random respectful recombination *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, 1991)* ed R K Belew and L B Booker (San Mateo, CA: Morgan Kaufmann) pp 222–9

Schaffer J D, Eshelman L J and Offutt D 1991 Spurious correlations and premature convergence in genetic algorithms *Foundations of Genetic Algorithms* ed G J E Rawlins (San Mateo, CA: Morgan Kaufmann) pp 102–12

Schraudolph N N and Belew R K 1992 Dynamic parameter encoding for genetic algorithms *Machine Learning* **9** 9–21

Shaefer C G 1987 The ARGOT strategy: adaptive representation genetic optimizer technique *Genetic Algorithms and Their Applications: Proc. 2nd Int. Conf. on Genetic Algorithms (Cambridge, MA, 1987)* ed J J Grefenstette (Hillsdale, NJ: Erlbaum) pp 50–8

Simon H A 1983 *Reason in Human Affairs* (Stanford, CA: Stanford University Press)

Spears W M and De Jong K A 1991 On the virtues of parameterized uniform crossover *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, 1991)* ed R K Belew and L B Booker (San Mateo, CA: Morgan Kaufmann) pp 230–6

Syswerda G 1989 Uniform crossover in genetic algorithms *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 2–9

——1991 Schedule optimization using genetic algorithms *Handbook of Genetic Algorithms* ed L Davis (New York: Van Nostrand Reinhold) pp 332–49

——1993 Simulated crossover in genetic algorithms *Foundations of Genetic Algorithms 2* ed D Whitley (San Mateo, CA: Morgan Kaufmann) pp 239–55

Whitley D 1989 The GENITOR algorithm and selection pressure: why rank-based allocation of reproductive trials is best *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 116–21

Whitley D, Starkweather T and Fuquay D 1989 Scheduling problems and traveling salesmen: the genetic edge recombination operator *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 116–21

Wright A 1991 Genetic algorithms for real parameter optimization *Foundations of Genetic Algorithms* ed G J E Rawlins (San Mateo, CA: Morgan Kaufmann) pp 205–18

B1.3 Evolution strategies

Günter Rudolph

Abstract

This section provides a description of evolution strategies (ESs) as a special instance of evolutionary algorithms. After the presentation of the archetype of ESs, accompanied with some historical remarks, contemporary ESs (the standard instances) are described in detail. Finally, the design and potential fields of application of nested ESs are discussed.

B1.3.1 The archetype of evolution strategies

Minimizing the total drag of three-dimensional slender bodies in a turbulent flow was, and still is, a general goal of research in institutes of hydrodynamics. Three students (Peter Bienert, Ingo Rechenberg, and Hans-Paul Schwefel) met each other at such an institute, the Hermann Föttinger Institute of the Technical University of Berlin, in 1964. Since they were amazed not only by aerodynamics, but also by cybernetics, they hit upon the idea to solve the analytically (and at that time also numerically) intractable form design problem with the help of some kind of robot. The robot should perform the necessary experiments by iteratively manipulating a flexible model positioned at the outlet of a wind tunnel. An *experimentum crucis* was set up with a two-dimensional foldable plate. The iterative search strategy—first performed by hand, a robot was developed later on by Peter Bienert—was expected to end up with a flat plate: the form with minimal drag. But it did not, since a one-variable-at-a-time as well as a discrete gradient-type strategy always got stuck in a local minimum: an S-shaped folding of the plate. Switching to small random changes that were only accepted in the case of improvements—an idea of Ingo Rechenberg—brought the breakthrough, which was reported at the joint annual meeting of WGLR and DGRR in Berlin, 1964 (Rechenberg 1965). The interpretation of binomially distributed changes as mutations and of the decision to step back or not as selection (on 12 June 1964) was the seed for all further developments leading to evolution strategies (ESs) as they are known today. So much about the birth of the *ES*.

It should be mentioned that the domain of the decision variables was not fixed or even restricted to real variables at that time. For example, the experimental optimization of the shape of a supersonic two-phase nozzle by means of mutation and selection required discrete variables and mutations (Klockgether and Schwefel 1970) whereas first numerical experiments with the early ES on a Zuse Z 23 computer (Schwefel 1965) employed discrete mutations of real variables. The apparent fixation of ESs to Euclidean search spaces nowadays is probably due to the fact that Rechenberg (1973) succeeded in analyzing the simple version in Euclidean space with continuous mutation for several test problems.

Within this setting the archetype of ESs takes the following form. An individual $\boldsymbol{a}$ consisting of an element $\boldsymbol{X} \in \mathbb{R}^n$ is mutated by adding a normally distributed random vector $\boldsymbol{Z} \sim N(0, \mathbf{I}_n)$ that is multiplied by a scalar $\sigma > 0$ ($\mathbf{I}_n$ denotes the unit matrix with rank n). The new point is accepted if it is better than or equal to the old one, otherwise the old point passes to the next iteration. The selection decision is based on a simple comparison of the objective function values of the old and the new point. Assuming that the objective function $f : \mathbb{R}^n \to \mathbb{R}$ is to be minimized, the simple ES, starting at some point $\boldsymbol{X}_0 \in \mathbb{R}^n$, is determined by the following iterative scheme:

$$\boldsymbol{X}_{t+1} = \begin{cases} \boldsymbol{X}_t + \sigma_t \boldsymbol{Z}_t & \text{if} \quad f(\boldsymbol{X}_t + \sigma_t \boldsymbol{Z}_t) \le f(\boldsymbol{X}_t) \\ \boldsymbol{X}_t & \text{otherwise} \end{cases} \tag{B1.3.1}$$

where $t \in \mathbb{N}_0$ denotes the iteration counter and where $(\boldsymbol{Z}_t : t \geq 0)$ is a sequence of independent and identically distributed standard normal random vectors.

The general algorithmic scheme (B1.3.1) was not a novelty: Schwefel (1995, pp 94–5), presents a survey of forerunners and related versions of (B1.3.1) since the late 1950s. Most methods differed in the mechanism to adjust the parameter σ_t, that is used to control the strength of the mutations (i.e. the length of the mutation steps in n-dimensional space). Rechenberg's solution to control parameter σ_t is known as the $1/5$ success rule: Increase σ_t if the relative frequency of successful mutations over some period in the past is larger than $1/5$, otherwise decrease σ_t. Schwefel (1995, p 112), proposed the following implementation. Let $t \in \mathbb{N}$ be the generation (or mutation) counter and assume that $t \geq 10n$.

(i) If $t \bmod n = 0$ then determine the number s of successful mutations that have occurred during the steps $t - 10n$ to $t - 1$.
(ii) If $s < 2n$ then multiply the step lengths by the factor 0.85.
(iii) If $s > 2n$ then divide the step lengths by the factor 0.85.

First ideas to extend the simple ES (B1.3.1) can be found in the book by Rechenberg (1973, pp 78–86). The population consists of $\mu > 1$ parents. Two parents are selected at random and recombined by multipoint crossover and the resulting individual is finally mutated. The offspring is added to the population. The selection operation chooses the μ best individuals out of the $\mu + 1$ in total to serve as parents of the next iteration. Since the search space was binary, this ES was exactly the same evolutionary algorithm as became known later under the term *steady-state genetic algorithm*. The usage of this algorithmic scheme C2.7.1 for Euclidean search spaces poses the problem of how to control the step length control parameter σ_t. Therefore, the 'steady-state' ES is no longer in use.

B1.3.2 Contemporary evolution strategies

The general algorithmic frame of contemporary ESs is easily presented by the symbolic notation introduced in Schwefel (1977). The abbreviation $(\mu + \lambda)$ ES denotes an ES that generates λ offspring from μ parents and selects the μ best individuals from the $\mu + \lambda$ individuals (parents and offspring) in total. This notation can be used to express the simple ES by $(1 + 1)$ ES and the 'steady-state' ES by $(\mu + 1)$ ES. Since the latter is not in use it is convention that the abbreviation $(\mu + \lambda)$ ES always refers to an ES parametrized according to the relation $1 \leq \mu \leq \lambda < \infty$.

The abbreviation (μ, λ) ES denotes an ES that generates λ offspring from μ parents but selects the μ best individuals only from the λ offspring. As a consequence, λ must be necessarily at least as large as μ. However, since the parameter setting $\mu = \lambda$ represents nothing more than a random walk, it is convention that the abbreviation (μ, λ) ES always refers to an ES parametrized according to the relation $1 \leq \mu < \lambda < \infty$.

Apart from the population concept contemporary ESs differ from early ESs in that an individual consists of an element $x \in \mathbb{R}^n$ of the search space plus several individual parameters controlling the individual mutation distribution. Usually, mutations are distributed according to a multivariate normal distribution with zero mean and some covariance matrix $\mathbf{C}$ that is symmetric and positive definite. Unless matrix $\mathbf{C}$ is a diagonal matrix, the mutations in each coordinate direction are *correlated* (Schwefel 1995, p 240). It was shown in Rudolph (1992) that a matrix is symmetric and positive definite if and only if it is decomposable via $\mathbf{C} = (\mathbf{ST})'\mathbf{ST}$ where $\mathbf{S}$ is a diagonal matrix with positive diagonal entries and

$$\mathbf{T} = \prod_{i=1}^{n-1} \prod_{j=i+1}^{n} \mathbf{R}_{ij}(\omega_k) \tag{B1.3.2}$$

is an orthogonal matrix built by a product of $n(n-1)/2$ elementary rotation matrices $\mathbf{R}_{ij}$ with angles $\omega_k \in (0, 2\pi]$. An elementary rotation matrix $\mathbf{R}_{ij}(\omega)$ is a unit matrix where four specific entries are replaced by $r_{ii} = r_{jj} = \cos\omega$ and $r_{ij} = -r_{ji} = -\sin\omega$.

As a consequence, $n(n-1)/2$ angles and n scaling parameters are sufficient to generate arbitrary correlated normal random vectors with zero mean and covariance matrix $\mathbf{C} = (\mathbf{ST})'\mathbf{ST}$ via $\boldsymbol{Z} = \mathbf{T}'\mathbf{S}'\boldsymbol{N}$, where $\boldsymbol{N}$ is a standard normal random vector (since matrix multiplication is associative, random vector $\boldsymbol{Z}$ can be created in $O(n^2)$ time by multiplication from right to left).

There remains, however, the question of how to choose and adjust these individual strategy parameters. The idea that a population-based ES could be able to adapt σ_t individually by including these parameters

in the mutation–selection process came up early (Rechenberg 1973, pp 132–7). Although first experiments with the $(\mu + 1)$ ES provided evidence that this approach works in principle, the first really successful implementation of the idea of *self-adaptation* was presented by Schwefel (1977) and it is based on the observation that a surplus of offspring (i.e. $\lambda > \mu$) is a good advice to establish self-adaptation of individual parameters. C7.1

To start with a simple case let $\mathbf{C} = \sigma^2 \mathbf{I}_n$. Thus, the only parameter to be self-adapted for each individual is the step length control parameter σ. For this purpose let the the genome of each individual be represented by the tuple $(\boldsymbol{X}, \sigma) \in \mathbb{R}^n \times \mathbb{R}_+$, that undergoes the genetic operators. Now mutation is a two-stage operation:

$$\sigma_{t+1} = \sigma_t \exp(\tau Z_\tau)$$
$$\boldsymbol{X}_{t+1} = \boldsymbol{X}_t + \sigma_{t+1} \boldsymbol{Z}$$

where $\tau = n^{-1/2}$ and Z_τ is a standard normal random variable whereas $\boldsymbol{Z}$ is a standard normal random vector. This scheme can be extended to the general case with $n\,(n+1)/2$ parameters.

(i) Let $\omega \in (0, 2\pi]^{n(n-1)/2}$ denote the angles that are necessary to build the orthogonal rotation matrix $\mathbf{T}(\omega)$ via (B1.3.2). The mutated angles $\omega_{t+1}^{(i)}$ are obtained by

$$\omega_{t+1}^{(i)} = (\omega_t^{(i)} + \varphi Z_\omega^{(i)}) \bmod 2\pi$$

where $\varphi > 0$ and the independent random numbers $Z_\omega^{(i)}$ with $i = 1, \ldots, n\,(n-1)/2$ are standard normally distributed.

(ii) Let $\sigma \in \mathbb{R}_+^n$ denote the standard deviations that are represented by the diagonal matrix $\mathbf{S}(\sigma) = \operatorname{diag}(\sigma^{(1)}, \ldots, \sigma^{(n)})$. The mutated standard deviations are obtained as follows. Draw a standard normally distributed random number Z_τ. For each $i = 1, \ldots, n$ set

$$\sigma_{t+1}^{(i)} = \sigma_t^{(i)} \exp(\tau Z_\tau + \eta Z_\sigma^{(i)})$$

where $(\tau, \eta) \in \mathbb{R}_+^2$ and the independent random numbers $Z_\sigma^{(i)}$ are standard normally distributed. Note that Z_τ is drawn only once.

(iii) Let $\boldsymbol{X} \in \mathbb{R}^n$ be the object variables and $\boldsymbol{Z}$ be a standard normal random vector. The mutated object variable vector is given by

$$\boldsymbol{X}_{t+1} = \boldsymbol{X}_t + \mathbf{T}(\omega_{t+1})\mathbf{S}(\sigma_{t+1})\boldsymbol{Z}.$$

According to Schwefel (1995) a good heuristic for the choice of the constants appearing in the above mutation operation is

$$(\varphi, \tau, \eta) = (5\pi/180,\ (2n)^{-1/2},\ (4n)^{-1/4})$$

but recent extensive simulation studies (Kursawe 1996) revealed that the above recommendation is not the best choice—especially in the case of multimodal objective functions it seems to be better to use weak selection pressure (μ/λ not too small) and a parametrization obeying the relation $\tau > \eta$. As a consequence, a final recommendation cannot be given here, yet.

As soon as $\mu > 1$, the decision variables as well as the internal strategy parameters can be recombined with usual recombination operators. Notice that there is no reason to employ the same recombination operator for the angles, standard deviations, and object variables. For example, one could apply *intermediate recombination* to the angles as well as standard deviations and *uniform crossover* to the object variables. With this choice recombination of two parents works as follows. Choose two parents $(\boldsymbol{X}, \sigma, \omega)$ and $(\boldsymbol{X}', \sigma', \omega')$ at random. Then the preliminary offspring resulting from the recombination process is C3.3

$$\left(\mathbf{U}\boldsymbol{X} + (\mathbf{I} - \mathbf{U})\boldsymbol{X}',\ \frac{\sigma + \sigma'}{2},\ \frac{(\omega + \omega') \bmod 4\pi}{2}\right)$$

where $\mathbf{I}$ is the unit matrix and $\mathbf{U}$ is a random diagonal matrix whose diagonal entries are either zero or one with the same probability. Note that the angles must be adjusted to the interval $(0, 2\pi]$.

After these preparations a sketch of a contemporary ES can be presented:

Generate μ initial parents of the type $(\boldsymbol{X}, \sigma, \omega)$ and determine their objective function values $f(\boldsymbol{X})$.
`repeat`
  `do` λ `times:`
    Choose $\rho \geq 2$ parents at random.
    Recombine their angles, standard deviations, and object variables.
    Mutate the angles, standard deviations, and object variables of the preliminary offspring obtained via recombination.
    Determine the offspring's objective function value.
    Put the offspring into the offspring population.
  `end do`
  Select the μ best individuals either from the offspring population
  or from the union of the parent and offspring population.
  The selected individuals represent the new parents.
`until` some stopping criterion is satisfied.

It should be noted that there are other proposals to adapt σ_t. In the case of a $(1, \lambda)$ ES with $\lambda = 3k$ and $k \in \mathbb{N}$, Rechenberg (1994), p 47, devised the following rule: Generate k offspring with σ_t, k offspring with $c\,\sigma_t$ and k offspring with σ_t / c for some $c > 0$ ($c = 1.3$ is recommended for $n \leq 100$, for larger n the constant c should decrease).

Further proposals, that are however still in an experimental state, try to derandomize the adaptation process by exploiting information gathered in preceding iterations (Ostermeier *et al* 1995). This approach is related to (deterministic) variable metric (or quasi-Newton) methods, where the Hessian matrix is approximated iteratively by certain update rules. The inverse of the Hessian matrix is in fact the optimal choice for the covariance matrix $\mathbf{C}$. A large variety of update rules is given by the *Oren–Luenberger class* (Oren and Luenberger 1974) and it might be useful to construct probabilistic versions of these update rules, but it should be kept in mind that ESs are designed to tackle difficult nonconvex problems and not convex ones: the usage of such techniques increases the risk that ESs will be attracted by local optima.

Other ideas that have not yet achieved a standard include the introduction of an additional age parameter κ for individuals in order to have intermediate forms of selection between the $(\mu + \lambda)$ ES with $\kappa = \infty$ and the (μ, λ) ES with $\kappa = 1$ (Schwefel and Rudolph 1995), as well as the huge variety of ESs whose population possesses a spatial structure. Since the latter is important for parallel implementations and applies to other evolutionary algorithms as well the description is omitted here.

B1.3.3 Nested evolution strategies

The shorthand notation $(\mu \dot{+} \lambda)$ ES was extended by Rechenberg (1978) to the expression

$$[\,\mu' \dot{+} \lambda'\,(\mu \dot{+} \lambda)^{\gamma}\,]^{\gamma'}\ \text{ES}$$

with the following meaning. There are μ' populations of μ parents. These are used to generate (e.g. by merging) λ' initial populations of μ individuals each. For each of these λ' populations a $(\mu \dot{+} \lambda)$ ES is run for γ generations. The criterion to rank the λ' populations after termination might be the average fitness of the individuals in each population. This scheme is repeated γ' times. The obvious generalization to higher levels of nesting is described by Rechenberg (1994), where it is also attempted to develop a shorthand notation to specify the parametrization completely.

This nesting technique is of course not limited to ESs: other evolutionary algorithms and even mixtures of them can be used instead. In fact, the somewhat artificial distinction between ESs, genetic algorithms, and evolutionary programs becomes more and more blurred when higher concepts enter the scene. Finally, some fields of application of nested evolutionary algorithms will be described briefly.

Alternative method to control internal parameters. Herdy (1992) used λ' subpopulations, each of them possessing its own different and fixed step size σ. Thus, there is no step size control at the level of individuals. After γ generations the improvements (in terms of fitness) achieved by each subpopulation is compared to each other and the best μ' subpopulations are selected. Then the process repeats with slightly modified values of σ. Since subpopulations with a near-optimal step size will achieve larger

improvements, they will be selected (i.e. better step sizes will survive), resulting in an alternative method to control the step size.

Mixed-integer optimization. Lohmann (1992) considered optimization problems in which the decision variables are partially discrete and partially continuous. The nested approach worked as follows. The ESs in the inner loop were optimizing over the continuous variables while the discrete variables were held fixed. After termination of the inner loop, the evolutionary algorithm in the outer loop compared the fitness values achieved in the subpopulations, selected the best ones, mutated the discrete variables and passed them as fixed parameters to the subpopulations in the inner loop.

It should be noted that this approach to mixed-integer optimization may cause some problems. In essence, a Gauß–Seidel-like optimization strategy is realized, because the search alternates between the subspace of discrete variables and the subspace of continuous variables. Such a strategy must fail whenever simultaneous changes in discrete *and* continuous variables are necessary to achieve further improvements.

Minimax optimization. Sebald and Schlenzig (1994) used nested optimization to tackle minimax problems of the type

$$\min_{x \in X}\{\max_{y \in Y}\{f(x, y)\}\}$$

where $X \subseteq \mathbb{R}^n$ and $Y \subseteq \mathbb{R}^m$. Equivalently, one may state the problem as follows:

$$\min\{g(x) : x \in X\} \qquad \text{where} \quad g(x) = \max\{f(x, y) : y \in Y\}.$$

The evolutionary algorithm in the inner loop maximizes $f(x, y)$ with fixed parameters x, while the outer loop is responsible to minimize $g(x)$ over the set X.

Other applications of this technique are imaginable. An additional aspect touches the evident degree of independence of executing the evolutionary algorithms in the inner loop. As a consequence, nested evolutionary algorithms are well suited for parallel computers.

References

Herdy M 1992 Reproductive isolation as strategy parameter in hierachically organized evolution strategies *Parallel Problem Solving from Nature, 2 (Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature, Brussels, 1992)* ed R Männer and B Manderick (Amsterdam: Elsevier) pp 207–17

Klockgether J and Schwefel H-P 1970 Two-phase nozzle and hollow core jet experiments *Proc. 11th Symp. on Engineering Aspects of Magnetohydrodynamics* ed D Elliott (Pasadena, CA: California Institute of Technology) pp 141–8

Kursawe F 1996 Breeding evolution strategies—first results, talk presented at Dagstuhl lectures *Applications of Evolutionary Algorithms (March 1996)*

Lohmann R 1992 Structure evolution and incomplete induction *Parallel Problem Solving from Nature, 2 (Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature, Brussels, 1992)* ed R Männer and B Manderick (Amsterdam: Elsevier) pp 175–85

Oren S and Luenberger D 1974 Self scaling variable metric (SSVM) algorithms, Part II: criteria and sufficient conditions for scaling a class of algorithms *Management Sci.* **20** 845–62

Ostermeier A, Gawelczyk A and Hansen N 1995 A derandomized approach to self-adaptation of evolution strategies *Evolut. Comput.* **2** 369–80

Rechenberg I 1965 *Cybernetic solution path of an experimental problem* Library Translation 1122, Royal Aircraft Establishment, Farnborough, UK

——1973 *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution* (Stuttgart: Frommann-Holzboog)

——1978 Evolutionsstrategien *Simulationsmethoden in der Medizin und Biologie* ed B Schneider and U Ranft (Berlin: Springer) pp 83–114

——1994 *Evolutionsstrategie '94* (Stuttgart: Frommann-Holzboog)

Rudolph G 1992 On correlated mutations in evolution strategies *Parallel Problem Solving from Nature, 2 (Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature, Brussels, 1992)* ed R Männer and B Manderick (Amsterdam: Elsevier) pp 105–14

Schwefel H-P 1965 *Kybernetische Evolution als Strategie der experimentellen Forschung in der Strömungstechnik* Diplomarbeit, Technical University of Berlin

——1977 *Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie* (Basel: Birkhäuser)

——1995 *Evolution and Optimum Seeking* (New York: Wiley)

Schwefel H-P and Rudolph G 1995 Contemporary evolution strategies *Advances in Artificial Life* ed F Morana *et al* (Berlin: Springer) pp 893–907

Sebald A V and Schlenzig J 1994 Minimax design of neural net controllers for highly uncertain plants *IEEE Trans. Neural Networks* **NN-5** 73–82

B1.4 Evolutionary programming

V William Porto

Abstract

This section describes the basic concepts of evolutionary programming (EP) as originally introduced by Fogel, with extensions by numerous other researchers. EP is distinguished from other forms of evolutionary computation, such as genetic algorithms, in that it simulates evolution emphasizing the phenotypic relationship between parent and offspring, rather than the genetic relationship. Emphasis is placed on the use of one or more mutation operations which generate diversity among the population of solutions while maintaining a high degree of correlation between parent and offspring behavior. Recent efforts in the areas of pattern recognition, system identification, parameter optimization, and automatic control are presented.

B1.4.1 Introduction

Evolutionary programming (EP) is one of a class of paradigms for simulating evolution which utilizes the concepts of *Darwinian evolution* to iteratively generate increasingly appropriate solutions (organisms) in light of a static or dynamically changing environment. This is in sharp contrast to earlier research into artificial intelligence research which largely centered on the search for simple heuristics. Instead of developing a (potentially) complex set of rules which were derived from human experts, EP evolves a set of solutions which exhibit optimal behavior with regard to an environment and desired payoff function. In a most general framework, EP may be considered an optimization technique wherein the algorithm iteratively optimizes behaviors, parameters, or other constructs. As in all optimization algorithms, it is important to note that the point of optimality is completely independent of the search algorithm, and is solely determined by the adaptive topography (i.e. response surface) (Atmar 1992). A2.1

In its standard form, the basic evolutionary program utilizes the four main components of all evolutionary computation (EC) algorithms: initialization, variation, evaluation (scoring), and selection. At the basis of this, as well as other EC algorithms, is the presumption that, at least in a statistical sense, learning is encoded phylogenically versus ontologically in each member of the population. 'Learning' is a byproduct of the evolutionary process as successful individuals are retained through stochastic trial and error. Variation (e.g. mutation) provides the means for moving solutions around on the search space, preventing entrapment in local minima. The evaluation function directly measures fitness, or equivalently the behavioral error, of each member in the population with regard to the environment. Finally, the selection process probabilistically culls suboptimal solutions from the population, providing an efficient method for searching the topography.

The basic EP algorithm starts with a population of trial solutions which are initialized by random, heuristic, or other appropriate means. The size of the population, μ, may range over a broadly distributed set, but is in general larger than one. Each of these trial solutions is evaluated with regard to the specified fitness function. After the creation of a population of initial solutions, each of the *parent* members is altered through application of a mutation process; in strict EP, recombination is not utilized. Each parent member i generates λ_i progeny which are replicated with a stochastic error mechanism (mutation). The fitness or behavioral error is assessed for all offspring solutions with the selection process performed by one of several general techniques including: (i) the best μ solutions are retained to become the parents for

the next generation (*elitist*), or (ii) μ of the best solutions are statistically retained (*tournament*), or (iii) *proportional-based selection*. In most applications, the size of the population remains constant, but there is no restriction in the general case. The process is halted when the solution reaches a predetermined quality, a specified number of iterations has been achieved, or some other criterion (e.g. sufficient convergence) stops the algorithm. C2.7.4, C2.3 C2.2

EP differs philosophically from other evolutionary computational techniques such as *genetic algorithms* (GAs) in a crucial manner. EP is a top-down versus bottom-up approach to optimization. It is important to note that (according to neo-Darwinism) selection operates only on the phenotypic expressions of a genotype; the underlying coding of the phenotype is only affected indirectly. The realization that a sum of optimal parts rarely leads to an optimal overall solution is key to this philosophical difference. GAs rely on the identification, combination, and survival of 'good' building blocks (schemata) iteratively combining to form larger 'better' building blocks. In a GA, the coding structure (genotype) is of primary importance as it contains the set of optimal building blocks discovered through successive iterations. The *building block hypothesis* is an implicit assumption that the fitness is a separable function of the parts of the genome. This successively iterated local optimization process is different from EP, which is an entirely global approach to optimization. Solutions (or organisms) in an EP algorithm are judged solely on their fitness with respect to the given environment. No attempt is made to partition credit to individual components of the solutions. In EP (and in *evolution strategies* (ESs)), the variation operator allows for simultaneous modification of all variables at the same time. Fitness, described in terms of the behavior of each population member, is evaluated directly, and is the sole basis for survival of an individual in the population. Thus, a crossover operation designed to recombine building blocks is not utilized in the general forms of EP. B1.2 B2.5.3 B1.3

B1.4.2 History

The *genesis* of EP was motivated by the desire to generate an alternative approach to artificial intelligence. Fogel (1962) conceived of using the simulation of evolution to develop artificial intelligence which did not rely on heuristics, but instead generated organisms of increasing intellect over time. Fogel (1964, Fogel *et al* 1966) made the observation that intelligent behavior requires the ability of an organism to make correct predictions within its environment, while being able to translate these predictions into a suitable response for a given goal. This early work focused on evolving *finite-state machines* (see the articles by Mealy (1955), and Moore (1957) for a discussion of these automata) which provided a most generic testbed for this approach. A finite-state machine (figure B1.4.1) is a mechanism which operates on a finite set (i.e. alphabet) of input symbols, possesses a finite number of internal states, and produces output symbols from a finite alphabet. As in all finite-state machines, the corresponding input–output symbol pairs and state transitions from every state define the specific behavior of the machine. A2.3.2 C1.5

In a series of experiments (Fogel *et al* 1966), an environment was simulated by a sequence of symbols from a finite-length alphabet. The problem was defined as follows: evolve an algorithm which would operate on the sequence of symbols previously observed in a manner that would produce an output symbol which maximizes the benefit to the algorithm in light of the next symbol to appear in the environment, relative to a well-defined payoff function.

EP was originally defined by Fogel (1964) in the following manner. A population of *parent* finite-state machines, after initialization, is exposed to the sequence of symbols (i.e. environment) which have been observed up to the current time. As each input symbol is presented to each parent machine, the output symbol is observed (predicted) and compared to the next input symbol. A predefined payoff function provides a means for measuring the worth of each prediction. After the last prediction is made, some function of the sequence of payoff values is used to indicate the overall fitness of each machine. Offspring machines are created by randomly mutating each parent machine. As defined above, there are five possible resulting modes of random mutation for a finite-state machine. These are: (i) change an output symbol; (ii) change a state transition; (iii) add a state; (iv) delete an existing state; and (v) change the initial state. Other mutations were proposed but results of experiments with these mutations were not described by Fogel *et al* (1966). To prevent the possibility of creating null machines, the deletion of a state and the changing of the initial state were allowed only when a parent machine had more than one state.

Mutation operators are chosen with respect to a specified probability distribution which may be uniform, or another desired distribution. The number of mutation operations applied to each offspring is also determined with respect to a specified probability distribution function (e.g. Poisson) or may be

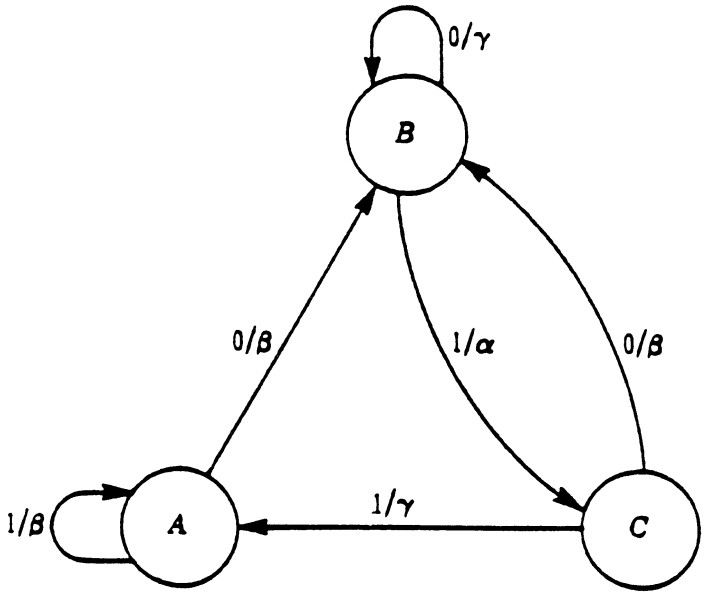


Figure B1.4.1. A simple finite-state machine diagram. Input symbols are shown to the left of the slash. Output symbols are to the right of the slash. The finite-state machine is presumed to start in state A.

fixed *a priori*. Each of the mutated offspring machines is evaluated over the existing environment (set of input–output symbol pairs) in the same manner as the parent machines.

After offspring have been created through application of the mutation operator(s) on the members of the parent population, the machines providing the greatest payoff with respect to the payoff function are retained to become parent members for the next generation. Typically, one offspring is created for each parent, and half of the total machines are retained in order to maintain a constant population size. The process is iterated until it is required to make an actual prediction of the next output symbol in the environment, which has yet to be encountered. This is analogous to the presentation of a naive exemplar to a previously trained neural network. Out of the entire population of machines, only the best machine, in terms of its overall worth, is chosen to generate the new output symbol. Fogel originally proposed selection of machines which score in the top half of the entire population, i.e. a nonregressive selection mechanism. Although discussed as a possibility to increase variance, the retention of lesser-quality machines was not incorporated in these early experiments.

Since the topography (response surface) is changed after each presentation of a symbol, the fitness of the evolved machines must change to reflect the payoff from the previous prediction. This prevents evolutionary stagnation as the adaptive topography is experiencing continuous change. As is evidenced in nature, the complexity of the representation is increased as the finite-state machines learn to recognize more subtle features in the experienced sequence of symbols.

B1.4.2.1 Early foundations

Fogel (see Fogel 1964, Fogel *et al* 1966) used EP on a series of successively more difficult prediction tasks. These experiments ranged from simple two-symbol cyclic sequences, eight-symbol cyclic sequences degraded by addition of noise, and sequences of symbols generated by other finite-state machines to nonstationary sequences and sequences taken from the article by Flood (1962).

In one example, the capability for predicting the 'primeness', i.e. whether or not a number is prime, was tested. A nonstationary sequence of symbols was generated by classifying each of the monotonically increasing set of integers as prime (with symbol 1) or nonprime (with symbol 0). The payoff function consisted of an all-or-none function where one point was provided for each correct prediction. No points or penalty were assessed for incorrect predictions. A small penalty term was added to maximize parsimony, through the subtraction of 0.01 multiplied by the number of states in the machine. This complexity penalty was added due to the limited memory available on the computers at that time. After presentation of 719 symbols, the iterative process was halted with the best machine possessing one state, with both

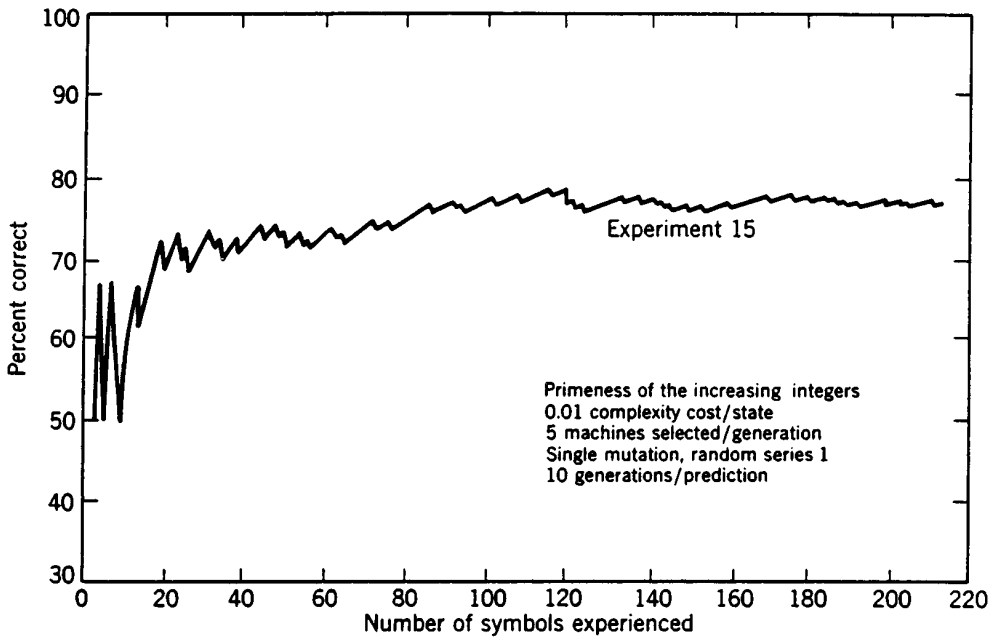


Figure B1.4.2. A plot showing the convergence of EP on finite-state machines evolved to predict primeness of numbers.

output symbols being zero. Figure B1.4.2 indicates the prediction score achieved in this nonstationary environment. Because prime numbers become increasingly infrequent (Burton 1976), the asymptotic worth of this machine, given the defined payoff function, approaches 100%.

After initial, albeit qualified, success with this experiment, the goal was altered to provide a greater payoff for correct prediction of a rare event. Correct prediction of a prime was worth one plus the number of nonprimes preceding it. For the first 150 symbols, 30 correct predictions were made (primes predicted as primes), 37 false positives (nonprimes predicted as primes), and five primes were missed. On predicting the 151st through 547th symbols there were 65 correct predictions of primes, and 67 false positives. Of the first 35 prime numbers, five were missed, but of the next 65 primes, none were missed. Fogel *et al* (1966) indicated that the machines demonstrated the capability to quickly recognize numbers which are divisible by two and three as being nonprime, and that some capability to recognize divisibility by five as being indicative of nonprimes was also evidenced. Thus, the machines generated evidence of learning a definition of primeness without prior knowledge of the explicit nature of a prime number, or any ability to explicitly divide.

Fogel and Burgin (1969) researched the use of EP in game theory. In a number of experiments, EP was consistently able to discover the globally optimal strategy in simple two-player, zero-sum games involving a small number of possible plays. This research also showed the ability of the technique to outperform human subjects in more complicated games. Several extensions were made to the simulations to address nonzero-sum games (e.g. pursuit evasion.) A three-dimensional model was constructed where EP was used to guide an *interceptor* towards a moving target. Since the target was, in most circumstances, allowed a greater degree of maneuverability, the success or failure of the interceptor was highly dependent upon the learned ability to predict the position of the target without *a priori* knowledge of the target's dynamics.

A different aspect of EP was researched by Walsh *et al* (1970) where EP was used for prediction as a precursor to automatic control. This research concentrated on decomposing a finite-state machine into submachines which could be executed in parallel to obtain the overall output of the evolved system. A primary goal of this research was to maximize parsimony in the evolving machines. In these experiments, finite-state machines containing seven and eight states were used as the generating function for three output symbols. The performance of three human subjects was compared to the evolved models when predicting the next symbol in the respective environments. In these experiments, EP was consistently able to outperform the human subjects.

B1.4.2.2 Extensions

The basic EP paradigm may be described by the following EP algorithm:

$t := 0;$
initialize $P(0) := \{a'_1(0), a'_2(0), \ldots, a'_\mu(0)\}$
evaluate $P(0) : \{\Phi(a'_1(0)), \Phi(a'_2(0)), \ldots, \Phi(a'_\mu(0))\}$
iterate
{
mutate: $P'(t) := m_{\Theta_m}(P(t))$
evaluate: $P'(t) : \{\Phi(a'_1(t)), \Phi(a'_2(t)), \ldots, \Phi(a'_\lambda(t))\}$
select: $P(t+1) := s_{\Theta_s}(P'(t) \cup Q)$
$t := t + 1;$
}

where:
a' is an individual member in the population
$\mu \geq 1$ is the size of the parent population
$\lambda \geq 1$ is the size of the offspring population
$P(t) := \{a'_1(t), a'_2(t), \ldots, a'_\mu(t)\}$ is the population at time t
$\Phi : I \rightarrow \Re$ is the fitness mapping
m_{Θ_m} is the mutation operator with controlling parameters Θ_m
s_{Θ_s} is the selection operator $\ni s_{\Theta_s} : \left(I^\lambda \cup I^{\mu+\lambda}\right) \rightarrow I^\mu$
$Q \in \{\emptyset, P(t)\}$ is a set of individuals additionally accounted for in the selection step, i.e. parent solutions.

Other than on initialization, the search space is generally unconstrained; constraints are utilized for generation and initialization of starting parent solutions. Constrained optimization may be addressed through imposition of the requirement that (i) the *mutation operator* is formulated to only generate legitimate solutions (often impossible) or (ii) a *penalty function* is applied to offspring mutations lying outside the constraint bounds in such a manner that they do not become part of the next generation. The objective function explicitly defines the fitness values which may be scaled to positive values (although this is not a requirement, it is sometimes performed to alter the range for ease of implementation).

C3.2.4 C5.2

In early versions of EP applied to continuous parameter optimization (Fogel 1992) the mutation operator is Gaussian with a zero mean and variance obtained for each component of the object variable vector as the square root of a linear transform of the fitness value $\varphi(\boldsymbol{x})$.

$$x_i(k+1) := x_i(k) + \sqrt{\beta_i(k)\varphi(x_i(k) + \gamma_i)} + N_i(0, 1)$$

where $\boldsymbol{x}(k)$ is the object variable vector, β is the proportionality constant, and γ is an offset parameter. Both β and γ must be set externally for each problem. $N_i(0, 1)$ is the ith independent sample from a Gaussian distribution with zero mean and unit variance.

Several extensions to the finite-state machine formulation of Fogel *et al* (1966) have been offered to address continuous optimization problems as well as to allow for various degrees of parametric self-adaptation (Fogel 1991a, 1992, 1995). There are three main variants of the basic paradigm, identified as follows:

(i) original EP, where continuous function optimization is performed without any self-adaptation mechanism;
(ii) *continuous* EP where new individuals in the population are inserted directly without iterative generational segmentation (i.e. an individual becomes part of the existing (surviving) population without waiting for the conclusion of a discrete generation; this is also known as *steady-state selection* in GAs and $(\mu + 1)$ *selection* in ES);
(iii) *self-adaptive* EP, which augments the solution vectors with one or more parameters governing the mutation process (e.g. variances, covariances) to permit self-adaptation of these parameters through the same iterative mutation, scoring, and selection process. In addition, self-adaptive EP may also be continuous in the sense of (ii) above.

C2.7.3 B1.3 C7.1

The original EP is an extension of the formulation of Fogel *et al* (1966) wherein continuous-valued functions replace the discrete alphabets of finite-state machines. The continuous form of EP was investigated by Fogel and Fogel (1993). To properly simulate this algorithmic variant, it is necessary to insert new population members by asynchronous methods (e.g. event-driven interrupts in a true multitasking, real-time operating system). Iterative algorithms running on a single central processing unit (CPU) are much more prevalent, since they are easier to program on today's computers, hence most implementations of EP are performed on a generational (epoch-to-epoch) basis.

Self-adaptive EP is an important extension of the algorithm in that it successfully overcomes the need for explicit user-tuning of the parameters associated with mutation. Global convergence may be obtained even in the presence of suboptimal parameterization, but available processing time is most often a precious resource and any mechanism for optimizing the convergence rate is helpful. As proposed by Fogel (1992, 1995), EP can self-adapt the variances for each individual in the following way:

$$x_i(k+1) := x_i(k) + \upsilon_i(k) * N_i(0,1)$$
$$\upsilon_i(k+1) := \upsilon_i(k) + [\sigma \upsilon_i(k)]^{1/2} * N_i(0,1).$$

The variable σ ensures that the variance υ_i remains nonnegative. Fogel (1992) suggests a simple rule wherein $\forall \upsilon_i(k) \leq 0$, $\upsilon_i(k)$ is set to ξ, a value close to but not identically equal to zero (to allow some degree of mutation). The sequence of updating the object variable x_i and variance υ_i was proposed to occur in opposite order from that of ESs (Bäck and Schwefel 1993, Rechenberg 1965, Schwefel 1981). Gehlhaar and Fogel (1996) provide evidence favoring the ordering commonly found in ES.

Further development of this theme led Fogel (1991a, 1992) to extend the procedure to alter the correlation coefficients between components of the object vector. A symmetric correlation coefficient matrix **P** is incorporated into the evolutionary paradigm in addition to the self-adaptation of the standard deviations. The components of **P** are initialized over the interval $[-1, 1]$ and mutated by perturbing each component, again, through the addition of independent realizations from a Gaussian random distribution. Bounding limits are placed upon the resultant mutated variables wherein any mutated coefficient which exceeds the bounds $[-1, 1]$ is reset to the upper or lower limit, respectively. Again, this methodology is similar to that of Schwefel (1981), as perturbations of both the standard deviations and rotation angles (determined by the covariance matrix **P**) allow adaptation to arbitrary contours on the error surface. This self- adaptation through the incorporation of correlated mutations across components of each parent object vector provides a mechanism for expediting the convergence rate of EP.

Fogel (1988) developed different selection operators which utilized *tournament competition* between solution organisms. These operators assigned a number of wins for each solution organism based on a set of individual competitions (using fitness scores as the determining factor) among each solution and each of the q competitors randomly selected from the total population. C2.3

B1.4.3 Current directions

Since the explosion of research into evolutionary algorithms in the late 1980s and early 1990s, EP has been applied to a wide range of problem domains with considerable success. Application areas in the current literature include training, construction, and optimization of neural networks, optimal routing (in two, three, and higher dimensions), drug design, bin packing, automatic control, game theory, and optimization of intelligently interactive behaviors of autonomous entities, among many others. Beginning in 1992, annual conferences on EP have brought much of this research into the open where these and other applications as well as basic research have expanded into numerous interdisciplinary realms.

Notable within a small sampling of the current research is the work in *neural network design*. Early efforts (Porto 1989, Fogel *et al* 1990, McDonnell 1992, and others) focused on utilizing EP for training neural networks to prevent entrapment in local minima. This research showed not only that EP was well suited to training a range of network topologies, but also that it was often more efficient than conventional (e.g. gradient-based) methods and was capable of finding optimal weight sets while escaping local minima points. Later research (Fogel 1992, Angeline *et al* 1994, McDonnell and Waagen 1993) involved simultaneous evolution of both the weights and structure of feedforward and feedback networks. Additional research into the areas of using EP for robustness training (Sebald and Fogel 1992), and for designing *fuzzy neural networks* for feature selection, pattern clustering, and classification (Brotherton and Simpson 1995) have been very successful as well as instructive. D1 D2

EP has been also used to solve optimal routing problems. The *traveling salesman problem* (TSP), one of many in the class of nondeterministic-polynomial-time- (NP-) complete (see Aho *et al* 1974) problems, has been studied extensively. Fogel (1988, 1993) demonstrated the capability of EP to address such problems. A representation was used wherein each of the cities to be visited was listed in order, with candidate solutions being permutations of this listing. A population of random tours is scored with respect to their Euclidean length. Each of the tours is mutated using one of many possible inversion operations (e.g. select two cities in the tour at random and reverse the order of the segment defined by the two cities) to generate an offspring. All of the offspring are then scored, with either elitist or stochastic competition used to cull lower-scoring members from the population. Optimization of the tours was quite rapid. In one such experiment with 1000 cities uniformly distributed, the best tour (after only 4×10^7 function evaluations) was estimated to be within 5–7% of the optimal tour length. Thus, excellent solutions were obtained after searching only an extremely small portion of the total potential search space. G9.5

EP has also been utilized in a number of medical applications. For example, the issue of optimizing drug design was researched by Gehlhaar *et al* (1995). EP was utilized to perform a conformational and position search within the binding site of a protein. The search space of small molecules which could potentially dock with the crystallographically determined binding site was explored iteratively guided by a database of crystallographic protein–ligand complexes. Geometries were constrained by known physical (in three dimensions) and chemical bounds. Results demonstrated the efficacy of this technique as it was orders of magnitude faster in finding suitable ligands than previous hands-on methodologies. The probability of successfully predicting the proper binding modes for these ligands was estimated at over 95% using nominal values for the crystallographic binding mode and number of docks attempted. These studies have permitted the rapid development of several candidate drugs which are currently in clinical trials.

The issue of utilizing EP to *control systems* has been addressed widely (Fogel and Fogel 1990, Fogel 1991a, Page *et al* 1992, and many others). Automatic control of fuzzy heating, ventilation, and air conditioning (HVAC) controllers was addressed by Haffner and Sebald (1993). In this study, a nonlinear, multiple-input, multiple-output (MIMO) model of a HVAC system was used and controlled by a fuzzy controller designed using EP. Typical fuzzy controllers often use trial and error methods to determine parameters and transfer functions, hence they can be quite time consuming with a complex MIMO HVAC system. These experiments used EP to design the membership functions and values (later studies were extended to find rules as well as responsibilities of the primary controller) to automate the tuning procedure. EP worked in an overall search space containing 76 parameters, 10 controller inputs, seven controller outputs, and 80 rules. Simulation results demonstrated that EP was quite effective at choosing the membership functions of the control laboratory and corridor pressures in the model. The synergy of combining EP with fuzzy set constructs proved quite fruitful in reducing the time required to design a stable, functioning HVAC system. F1.3

Game theory has always been at the forefront of artificial intelligence research. One interesting game, the iterated prisoner's dilemma, has been studied by numerous investigators (Axelrod 1987, Fogel 1991b, Harrald and Fogel 1996, and others). In this two-person game, each player can choose one of two possible behavioral policies: defection or cooperation. Defection implies increasing one's own reward at the expense of the opposing player, while cooperation implies increasing the reward for both players. If the game is played over a single iteration, the dominant move is defection. If the players' strategies depend on the results of previous iterations, mutual cooperation may possibly become a rational policy, whereas if the sequence of strategies is not correlated, the game degenerates into a series of single plays with defection being the end result. Each player must choose to defect or cooperate on each trial. Table B1.4.1 describes a general form of the payoff function in the prisoner's dilemma.

In addition, the payoff matrix defining the game is subject to the following constraints (Rapoport 1966):

$$2\gamma_1 > \gamma_2 + \gamma_3$$
$$\gamma_3 > \gamma_1 > \gamma_4 > \gamma_2.$$

Both neural network approaches (Harrald and Fogel 1996) and finite-state machine approaches (Fogel 1991b) have been applied to this problem. Finite-state machines are typically used where there are discrete choices between cooperation and defection. Neural networks allow for a continuous range of choices between these two opposite strategies. Results of these preliminary experiments using EP, in

Table B1.4.1. A general form of the payoff matrix for the prisoner's dilemma problem. γ_1 is the payoff to each player for mutual cooperation. γ_2 is the payoff for cooperating when the other player defects. γ_3 is the payoff for defecting when the other player cooperates. γ_4 is the payoff to each player for mutual defection. Entries (α, β) indicate payoffs to players A and B, respectively.

		Player B	
		C	D
Player A	C	(γ_1, γ_1)	(γ_2, γ_3)
	D	(γ_3, γ_2)	(γ_4, γ_4)

general, indicated that mutual cooperation is more likely to occur when the behaviors are limited to the extremes (the finite-state machine representation of the problem), whereas in the neural network continuum behavioral representation of the problem, it is easier to slip into a state of mutual defection.

Development of interactively intelligent behaviors was investigated by Fogel *et al* (1996). EP was used to optimize computer-generated force (CGF) behaviors such that they learned new courses of action adaptively as changes in the environment (i.e. presence or absence of opposing side forces) were encountered. The actions of the CGFs were created at the response of an event scheduler which recognized significant changes in the environment as perceived by the forces under evolution. New plans of action were found during these event periods by invoking an evolutionary program. The iterative EP process was stopped when time or CPU limits were met, and relinquished control of the simulated forces back to the CGF simulator after transmitting newly evolved instruction sets for each simulated unit. This process proved quite successful and offered a significant improvement over other rule-based systems.

B1.4.4 Future research

Important research is currently being conducted into the understanding of the *convergence properties* of EP, B2.2.5 as well as the basic mechanisms of different mutation operators and selection mechanisms. Certainly of great interest is the potential for self-adaptation of exogeneous parameters of the mutation operation (meta and Rmeta-EP) , as this not only frees the user from the often difficult task of parameterization, but also provides a built-in, automated mechanism for providing optimal settings throughout a range of problem domains. The number of application areas of this optimization technique is constantly growing. EP, along with the other EC techniques, is being used on previously untenable, often NP-complete, problems which occur quite often in commercial and military problems.

References

Aho A V, Hopcroft J E and Ullman J D 1974 *The Design and Analysis of Computer Algorithms* (Reading, MA: Addison-Wesley) pp 143–5, 318–26

Angeline P, Saunders G and Pollack J 1994 Complete induction of recurrent neural networks *Proc. 3rd Ann. Conf. on Evolutionary Programming (San Diego, CA, 1994)* ed A V Sebald and L J Fogel (Singapore: World Scientific) pp 1–8

Atmar W 1992 On the rules and nature of simulated evolutionary programming *Proc. 1st Ann. Conf. on Evolutionary Programming (La Jolla, CA, 1992)* ed D B Fogel and W Atmar (San Diego, CA: Evolutionary Programming Society) pp 17–26

Axelrod R 1987 The evolution of strategies in the iterated prisoner's dilemma *Genetic Algorithms and Simulated Annealing* ed L Davis (London) pp 32–42

Bäck T and Schwefel H-P 1993 An overview of evolutionary algorithms for parameter optimization *Evolutionary Comput.* **1** 1–23

Brotherton T W and Simpson P K 1995 Dynamic feature set training of neural networks for classification *Evolutionary Programming IV: Proc. 4th Ann. Conf. on Evolutionary Programming (San Diego, CA, 1995)* ed J R McDonnell, R G Reynolds and D B Fogel (Cambridge, MA: MIT Press) pp 83–94

Burton D M 1976 *Elementary Number Theory* (Boston, MA: Allyn and Bacon) p 136–52

Flood M M 1962 Stochastic learning theory applied to choice experiments with cats, dogs and men *Behavioral Sci.* **7** 289–314

Fogel D B 1988 An evolutionary approach to the traveling salesman problem *Biol. Cybernet.* **60** 139–44

——1991a *System Identification through Simulated Evolution* (Needham, MA: Ginn)

——1991b The evolution of intelligent decision making in gaming *Cybernet. Syst.* **22** 223–36
——1992 *Evolving Artificial Intelligence* PhD Dissertation, University of California
——1993 Applying evolutionary programming to selected traveling salesman problems *Cybernet. Syst.* **24** 27–36
——1995 *Evolutionary Computation, Toward a New Philosophy of Machine Intelligence* (Piscataway, NJ: IEEE)
Fogel D B and Fogel L J 1990 Optimal routing of multiple autonomous underwater vehicles through evolutionary programming *Proc. Symp. on Autonomous Underwater Vehicle Technology* (Washington, DC: IEEE Oceanic Engineering Society) pp 44–7
Fogel D B, Fogel L J and Porto V W 1990 Evolving neural networks *Biol. Cybernet.* **63** 487–93
Fogel G B and Fogel D B 1993 Continuous evolutionary programming: analysis and experiments *Cybernet. Syst.* **26** 79–90
Fogel L J 1962 Autonomous automata *Industrial Res.* **4** 14–9
Fogel L J 1964 *On The Organization of Intellect* PhD Dissertation, University of California
Fogel L J and Burgin G H 1969 *Competitive Goal-Seeking through Evolutionary Programming* Air Force Cambridge Research Labratories Final Report Contract AF19(628)-5927
Fogel L J, Owens A J and Walsh M J 1966 *Artificial Intelligence through Simulated Evolution* (New York: Wiley)
Fogel L J, Porto V W and Owen M 1996 An intelligently interactive non-rule-based computer generated force *Proc. 6th Conf. on Computer Generated Forces and Behavioral Representation* (Orlando, FL: Institute for Simulation and Training STRICOM-DMSO) pp 265–70
Gehlhaar D K and Fogel D B 1996 Tuning evolutionary programming for conformationally flexible molecular docking *Proc. 5th Ann. Conf. on Evolutionary Programming (1996)* ed L J Fogel, P J Angeline and T Bäck (Cambridge, MA: MIT Press) pp 419–29
Gehlhaar D K, Verkhivker G, Rejto P A, Fogel D B, Fogel L J and Freer S T 1995 Docking conformationally flexible small molecules into a protein binding site through evolutionary programming *Evolutionary Programming IV: Proc. 4th Ann. Conf. on Evolutionary Programming (San Diego, CA, 1995)* (Cambridge, MA: MIT Press) pp 615–27
Harrald P G and Fogel D B 1996 Evolving continuous behaviors in the iterated prisoner's dilemma *BioSystems* **37** 135–45
Haffner S B and Sebald A V 1993 Computer-aided design of fuzzy HVAC controllers using evolutionary programming *Proc. 2nd Ann. Conf. on Evolutionary Programming (San Diego, CA, 1993)* ed D B Fogel and W Atmar (San Diego, CA: Evolutionary Programming Society) pp 98–107
McDonnell J M 1992 Training neural networks with weight constraints *Proc. 1st Ann. Conf. on Evolutionary Programming (La Jolla, CA, 1992)* ed D B Fogel and W Atmar (San Diego, CA: Evolutionary Programming Society) pp 111–9
McDonnell J M and Waagen D 1993 Neural network structure design by evolutionary programming *Proc. 2nd Ann. Conf. on Evolutionary Programming (San Diego, CA, 1993)* ed D B Fogel and W Atmar (San Diego, CA: Evolutionary Programming Society) pp 79–89
Mealy G H 1955 A method of synthesizing sequential circuits *Bell Syst. Tech. J.* **34** 1054–79
Moore E F 1957 Gedanken-experiments on sequential machines: automata studies *Annals of Mathematical Studies* vol 34 (Princeton, NJ: Princeton University Press) pp 129–53
Page W C, McDonnell J M and Anderson B 1992 An evolutionary programming approach to multi-dimensional path planning *Proc. 1st Ann. Conf. on Evolutionary Programming (La Jolla, CA, 1992)* ed D B Fogel and W Atmar (San Diego, CA: Evolutionary Programming Society) pp 63–70
Porto V W 1989 Evolutionary methods for training neural networks for underwater pattern classification *24th Ann. Asilomar Conf. on Signals, Systems and Computers* vol 2 pp 1015–19
Rapoport A 1966 *Optimal Policies for the Prisoner's Dilemma* University of North Carolina Psychometric Laboratory Technical Report 50
Rechenberg I 1965 *Cybernetic Solution Path of an Experimental Problem* Royal Aircraft Establishment Translation 1122
Schwefel H-P 1981 *Numerical Optimization of Computer Models* (Chichester: Wiley)
Sebald A V and Fogel D B 1992 Design of fault tolerant neural networks for pattern classification *Proc. 1st Ann. Conf. on Evolutionary Programming (La Jolla, CA, 1992)* ed D B Fogel and W Atmar (San Diego, CA: Evolutionary Programming Society) pp 90–9
Walsh M J, Burgin G H and Fogel L J 1970 *Prediction and Control through the Use of Automata and their Evolution* US Navy Final Report Contract N00014-66-C-0284

Further reading

There are several excellent general references available to the reader interested in furthering his or her knowledge in this exciting area of EC. The following books are a few well-written examples providing a good theoretical background in EP as well as other evolutionary algorithms.

1. Bäck T 1996 *Evolutionary Algorithms in Theory and Practice* (New York: Oxford University Press)
2. Fogel D B 1995 *Evolutionary Computation, Toward a New Philosophy of Machine Intelligence* (Piscataway, NJ: IEEE)
3. Schwefel H-P 1981 *Numerical Optimization of Computer Models* (Chichester: Wiley)
4. Schwefel H-P 1995 *Evolution and Optimization Seeking* (New York: Wiley)

B1.5 Derivative methods

Kenneth E Kinnear, Jr (B1.5.1), *Robert E Smith* (B1.5.2)
and Zbigniew Michalewicz (B1.5.3)

Abstract

See the individual abstracts for sections B1.5.1–B1.5.3.

B1.5.1 Genetic programming

Kenneth E Kinnear, Jr

Abstract

The fundamental concepts of genetic programming are discussed here. Genetic programming is a form of evolutionary algorithm that is distinguished by a particular set of choices as to representation, genetic operator design, and fitness evaluation.

B1.5.1.1 Introduction

This article describes the fundamental concepts of genetic programming (GP) (Koza 1989, 1992). Genetic programming is a form of evolutionary algorithm which is distinguished by a particular set of choices as to representation, genetic operator design, and fitness evaluation. When examined in isolation, these choices define an approach to evolutionary computation (EC) which is considered by some to be a specialization of the *genetic algorithm* (GA). When considered together, however, these choices define a conceptually different approach to evolutionary computation which leads researchers to explore new and fruitful lines of research and practical applications. B1.2

B1.5.1.2 Genetic programming defined and explained

Genetic programming is implemented as an evolutionary algorithm in which the data structures that undergo adaptation are executable computer programs. Fitness evaluation in genetic programming involves executing these evolved programs. Genetic programming, then, involves an evolution-directed search of the space of possible computer programs for ones which, when executed, will produce the best fitness.

In short, genetic programming breeds computer programs. To create the initial population a large number of computer programs are generated at random. Each of these programs is executed and the results of that execution are used to assign a fitness value to each program. Then a new population of programs, the next generation, is created by directly copying certain selected existing programs, where the selection is based on their fitness values. This population is filled out by creating a number of new offspring programs through genetic operations on existing parent programs which are selected based, again, on their fitness. Then, this new population of programs is again evaluated and a fitness is assigned to each program based on the results of its evaluation. Eventually this process is terminated by the creation and evaluation of a 'correct' program or the recognition of some other specific termination criteria.

More specifically, at the most basic level, genetic programming is defined as a genetic algorithm with some unusual choices made as to the representation of the problem, the genetic operators used to modify that representation, and the fitness evaluation techniques employed.

A specialized representation: executable programs. Any evolutionary algorithm is distinguished by the structures used to represent the problem to be solved. These are the structures which undergo transformation, and in which the potential solutions reside.

Originally, most genetic algorithms used *linear strings of bits* as the structures which evolved (Holland 1975), and the representation of the problem was typically the encoding of these bits as numeric or logical parameters of a variety of algorithms. The evolving structures were often used as parameters to human-coded algorithms. In addition, the bitstrings used were frequently of fixed length, which aided in the translation into parameters for the algorithms involved. C1.2

More recently, genetic algorithms have appeared with real-valued numeric sequences used as the evolvable structures, still frequently used as parameters to particular algorithms. In recent years, many genetic algorithm implementations have appeared with sequences which are of variable length, sometimes based on the order of the sequences, and which contain more complex and structured information than parameters to existing algorithms.

The representation used by genetic programming is that of an executable program. There is no single form of executable program which is used by all genetic programming implementations, although many implementations use a *tree-structured representation* highly reminiscent of a LISP functional expression. These representations are almost always of a variable size, though for implementation purposes a maximum size is usually specified. C1.6

Figure B1.5.1 shows an example of a tree-structured representation for a genetic programming implementation. The specific task for which this is a reasonable representation is the learning of a Boolean function from a set of inputs.

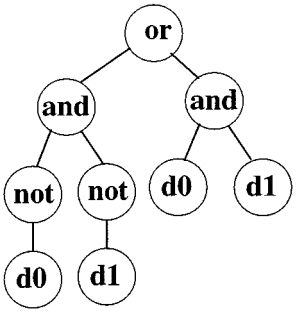


Figure B1.5.1. Tree-structured representation used in genetic programming.

This figure contains two different types of node (as do most genetic programming representations) which are called functions and terminals. Terminals are usually inputs to the program, although they may also be constants. They are the variables which are set to values external to the program itself prior to the fitness evaluation performed by executing the program. In this example d0 and d1 are the terminals. They can take on binary values of either zero or one.

Functions take inputs and produce outputs and possibly produce side-effects. The inputs can be either a terminal or the output of another function. In the above example, the functions are AND, OR, and NOT. Two of these functions are functions of two inputs, and one is a function of one input. Each produces a single output and no side effect.

The fitness evaluation for this particular *individual* is determined by the effectiveness with which it will produce the correct logical output for all of the test cases against which it is tested.

One way to characterize the design of a representation for an application of genetic programming to a particular problem is to view it as the design of a language, and this can be a useful point of view. Perhaps it is more useful, however, to view the design of a genetic programming representation as that of the design of a virtual machine—since usually the execution engine must be designed and constructed as well as the representation or language that is executed.

The representation for the program (i.e. the definition of the functions and terminals) must be designed along with the virtual machine that is to execute them. Rarely are the programs evolved in genetic programming given direct control of the central processor of a computer (although see the article by Nordin (1994)). Usually, these programs are interpreted under control of a virtual machine which defines the functions and terminals. This includes the functions which process the data, the terminals that provide

the inputs to the programs, and any control functions whose purpose is to affect the execution flow of the program.

As part of this virtual machine design task, it is important to note that the output of any function or the value of any terminal may be used as the input to any function. Initially, this often seems to be a trivial problem, but when actually performing the design of the representation and virtual machine to execute that representation, it frequently looms rather large. Two solutions are typically used for this problem. One approach is to design the virtual machine, represented by the choice of functions and terminals, to use only a single data type. In this way, the output of any function or the value of any terminal is acceptable as input to any function. A second approach is to allow more than one data type to exist in the virtual machine. Each function must then be defined to operate on any of the existing data types. Implicit coercions are performed by each function on its input to convert the data type that it receives to one that it is more normally defined to process. Even after handling the data type problem, functions must be defined over the entire possible range of argument values. Simple arithmetic division must be defined to return some value even when division by zero is attempted.

It is important to note that the definition of functions and the virtual machine that executes them is not restricted to functions whose only action is to provide a single output value based on their inputs. Genetic programming functions are often defined whose primary purpose is the actions they take by virtue of their side-effects. These functions must return some value as well, but their real purpose is interaction with an environment external to the genetic programming system.

An additional type of side-effect producing function is one that implements a control structure within the virtual machine defined to execute the genetically evolved program. All of the common programming control constructs such as if–then–else, while–do, for, and others have been implemented as evolvable control constructs within genetic programming systems. Looping constructs must be protected in such a way that they will never loop forever, and usually have an arbitrary limit set on the number of loops which they will execute.

As part of the initialization of a genetic programming run, a large number of individual programs are generated at random. This is relatively straightforward, since the genetic programming system is supplied with information about the number of arguments required by each function, as well as all of the available terminals. Random program trees are generated using this information, typically of a relatively small size. The program trees will tend to grow quickly to be quite large in the absence of some explicit evolutionary pressure toward small size or some simple hard-coded limits to growth (see Section C4.4 for some methods to handle this problem). C4.4

Genetic operators for evolving programs. The second specific design approach that distinguishes genetic programming from other types of genetic algorithm is the design of the genetic operators. Having decided to represent the problem to be solved as a population of computer programs, the essence of an evolutionary algorithm is to evaluate the fitness of the individuals in the population and then to create new members of the population based in some way on the individuals which have the highest fitness in the current population.

In genetic algorithms, *recombination* is typically the key genetic operator employed, with some utility ascribed to mutation as well. In this way, genetic programming is no different from any other genetic algorithm. genetic algorithms usually have genetic material organized in a linear fashion and the recombination, or crossover, algorithm defined for such genetic material is quite straightforward (see Section C3.3.1). The usual representation of genetic programming programs as tree-structured combinations of functions and terminals requires a different form of recombination algorithm. A major step in the invention of genetic programming was the design of a recombination operator which would simply and easily allow the creation of an offspring program tree using as inputs the program trees of two individuals of generally high fitness as parents (Cramer 1985, Koza 1989, 1992). C3.3.1

In any evolutionary algorithm it is vitally important that the fitness of the offspring be related to that of the parents, or else the process degenerates into one of random search across whatever representation space was chosen. It is equally vital that some variation, indeed heritable variation, be introduced into the offspring's fitness, otherwise no improvement toward an optimum is possible.

The tree-structured genetic material usually used in genetic programming has a particularly elegant recombination operator that may be defined for it. In figure B1.5.2, there are two parent program trees, (*a*) and (*b*). They are to be recombined through crossover to create an offspring program tree (*c*). A

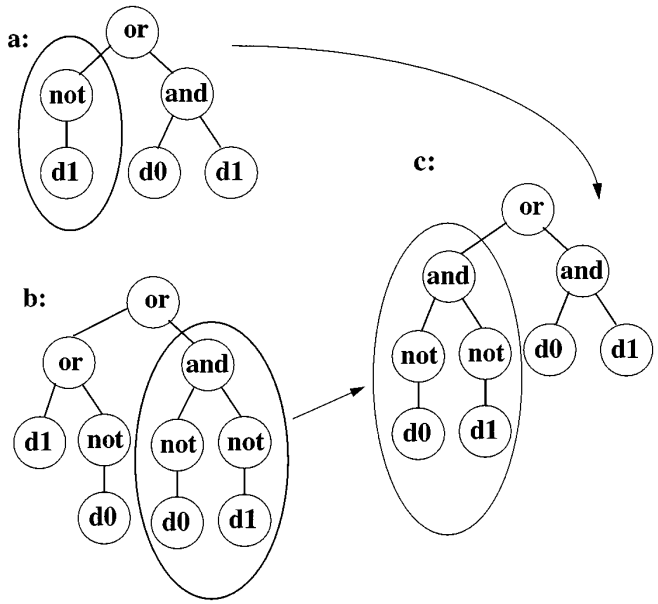


Figure B1.5.2. Recombination in genetic programming.

subtree is chosen in each of the parents, and the offspring is created by inserting the subtree chosen from (*b*) into the place where the subtree was chosen in (*a*). This very simply creates an offspring program tree which preserves the same constraints concerning the number of inputs to each function as each parent tree. In practice it yields a offspring tree whose fitness has enough relationship to that of its parents to support the evolutionary search process. Variations in this crossover approach are easy to imagine, and are currently the subject of considerable active research in the genetic programming community (D'haeseleer 1994, Teller 1996).

Mutation is a genetic operator which can be applied to a single parent program tree to create an offspring tree. The typical mutation operator used selects a point inside a parent tree, and generates a new random subtree to replace the selected subtree. This random subtree is usually generated by the same procedure used to generate the initial population of program trees. C3.2

Fitness evaluation of genetically evolved programs. Finally, then, the last detailed distinction between genetic programming and a more usual implementation of the genetic algorithm is that of the assignment of a fitness value for a individual.

In genetic programming, the representation of the individual is a program which, when executed under control of a defined virtual machine, implements some algorithm. It may do this by returning some value (as would be the case for a system to learn a specific Boolean function) or it might do this through the performance of some task through the use of functions which have side-effects that act on a simulated (or even the real) world.

The results of the program's execution are evaluated in some way, and this evaluation represents the fitness of the individual. This fitness is used to drive the selection process for copying into the next generation or for the selection of parents to undergo genetic operations yielding offspring. Any selection operator from those presented in Chapter C2 can be used. C2

There is certainly a desire to evolve programs using genetic programming that are 'general', that is to say that they will not only correctly process the fitness cases on which they are evolved, but will process correctly any fitness cases which could be presented to them. Clearly, in the cases where there are infinitely many possible cases, such as evolving a general sorting algorithm (Kinnear 1993), the evolutionary process can only be driven by a very limited number of fitness cases. Many of the lessons from machine learning on the tradeoffs between generality and performance on training cases have been helpful to genetic programming researchers, particularly those from decision tree approaches to machine learning (Iba *et al* 1994).

B1.5.1.3 The development of genetic programming

LISP was the language in which the ideas which led to genetic programming were first developed (Cramer 1985, Koza 1989, 1992). LISP has always been one of the preeminent language choices for implementation where programs need to be treated as data. In this case, programs are data while they are being evolved, and are only considered executable when they are undergoing fitness evaluation.

As genetic programming itself evolved in LISP, the programs that were executed began to look less and less like LISP programs. They continued to be tree structured but soon few if any of the functions used in the evolved programs were standard LISP functions. Around 1992 many people implemented genetic programming systems in C and C++, along with many other programming languages. Today, other than a frequent habit of printing the representation of tree-structured genetic programs in a LISP-like syntax, there is no particular connection between genetic programming and LISP.

There are many public domain implementations of genetic programming in a wide variety of programming languages. For further details, see the reading list at the end of this section.

B1.5.1.4 The value of genetic programming

Genetic programming is defined as a variation on the theme of genetic algorithms through some specific selections of representation, genetic operators appropriate to that representation, and fitness evaluation as execution of that representation in a virtual machine. Taken in isolation, these three elements do not capture the value or promise of genetic programming. What makes genetic programming interesting is the conceptual shift of the problem being solved by the genetic algorithm. A genetic algorithm searches for something, and genetic programming shifts the search from that of parameter discovery for some existing algorithm designed to solve a problem to a search for a program (or algorithm) to solve the problem directly. This shift has a number of ramifications.

- This conceptualization of evolving computer programs is powerful in part because it can change the way that we think about solving problems. Through experience, it has become natural to think about solving problems through a process of human-oriented program discovery. Genetic programming allows us to join this approach to problem solving with powerful EC-based search techniques.

  An example of this is a variation of genetic programming called stack genetic programming (Perkis 1994), where the program is a variable-length linear string of functions and terminals, and the argument passing is defined to be on a stack. The genetic operators in a linear system such as this are much closer to the traditional genetic algorithm operators, but the execution and fitness evaluation (possibly including side-effects) is equivalent to any other sort of genetic programming. The characteristics of stack genetic programming have not yet been well explored but it is clear that it has rather different strengths and weaknesses than does traditional genetic programming.

  Many of the approaches to simulation of adaptive behavior involve simple programs designed to control *animats*. The conceptualization of evolving computer programs as presented by genetic programming fits well with work on evolving adaptive entities (Reynolds 1994, Sims 1994).
- There has been a realization that not only can we evolve programs that are built from human-created functions and terminals, but that the functions from which they are built can evolve as well. Koza's invention of automatically defined functions (ADFs) (Koza 1994) is one such example of this realization. ADFs allow the definitions of certain subfunctions to evolve even while the functions that call them are evolving. For certain classes of problems, ADFs result in considerable increases in performance (Koza 1994, Angeline and Pollack 1993, Kinnear 1994).
- Genetic programming is capable of integrating a wide variety of existing capabilities, and has potential to tie together several complementary subsystems into an overall hybrid system. The functions need not be simple arithmetic or logical operators, but could instead be fast Fourier transforms, GMDH systems, or other complex building blocks. They could even be the results of other evolutionary computation algorithms.
- The genetic operators that create offspring programs from parent programs are themselves programs. These programs can also be evolved either as part of a separate process, or in a coevolutionary way with the programs on which they operate. While any evolutionary computation algorithm could have parameters that affect the genetic operators be part of the evolutionary process, genetic programming provides a natural way to let the operators (defined as programs) evolve directly (Teller 1996, Angeline 1996).

- Genetic programming naturally enhances the possibility for increasingly indirect evolution. As an example of the possibilities, genetic programming has been used to evolve grammars which, when executed, produce the structure of an untrained *neural network*. These neural networks are then trained, and the trained networks are then evaluated on a test set. The results of this evaluation are then used as the fitnesses of the evolved grammars (Gruau 1993). D1

This last example is a step along the path toward modeling embryonic development in genetic programming. The opportunity exists to evolve programs whose results are themselves programs. These resulting programs are then executed and their values or side-effects are evaluated—and become the fitness for the original evolving, program creating programs. The analogy to natural embryonic development is clear, where the genetic material, the genotype, produces through development a body, the phenotype, which then either does or does not produce offspring, the fitness (Kauffman 1993).

Genetic programming is valuable in part because we find it natural to examine issues such as those mentioned above when we think about evolutionary computation from the genetic programming perspective.

B1.5.2 Learning classifier systems

Robert E Smith

Abstract

Learning classifier systems (LCSs) are rule-based machine learning systems that use genetic algorithms as their primary rule discovery mechanism. There is no standard version of the LCS; however, all LCSs share some common characteristics. These characteristics are introduced here by first examining reinforcement learning problems, which are the most frequent application area for LCSs. This introduction provides a motivational framework for the representation and mechanics of LCSs. Also examined are the variations of the LCS scheme.

B1.5.2.1 Introduction

The learning classifier system (LCS) (Goldberg 1989, Holland *et al* 1986) is often referred to as the primary machine learning technique that employs genetic algorithms (GAs). It is also often described as a production system framework with a genetic algorithm as the primary rule discovery method. However, the details of LCS operation vary widely from one implementation to another. In fact, no standard version of the LCS exists. In many ways, the LCS is more of a concept than an algorithm. To explain details of the LCS concept, this article will begin by introducing the type of machine learning problem most often associated with the LCS. This discussion will be followed by a overview of the LCS, in its most common form. Final sections will introduce the more complex issues involved in LCSs.

B1.5.2.2 Types of learning problem

To introduce the LCS, it will be useful to describe types of machine learning problem. Often, in the literature, machine learning problems are described in terms of cognitive psychology or animal behavior. This discussion will attempt to relate the terms used in machine learning to engineering control.

Consider the generic control problem shown in figure B1.5.3. In this problem, inputs from an external control system, combined with uncontrollable disturbances from other sources, change the state of the plant. These changes in state are reflected in the state information provided by the plant. Note that, in general, the state information can be incomplete and noisy.

Consider the *supervised learning* problem shown in figure B1.5.4 (Barto 1990). In this problem, an inverse plant model (or teacher) is available that provides errors directly in terms of control actions. Given this direct error feedback, the parameters of the control system can be adjusted by means of gradient descent, to minimize error in control actions. Note that this is the method used in the neural network backpropagation algorithm.

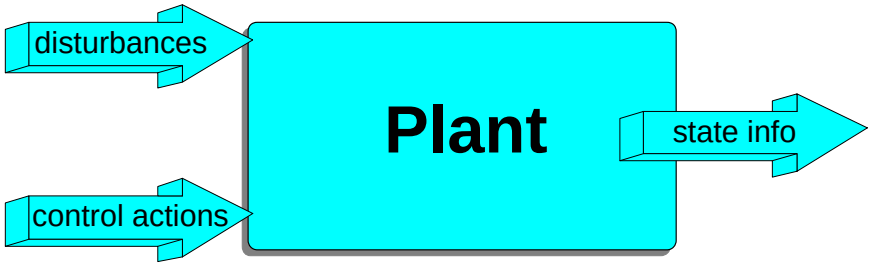


Figure B1.5.3. A generic control problem.

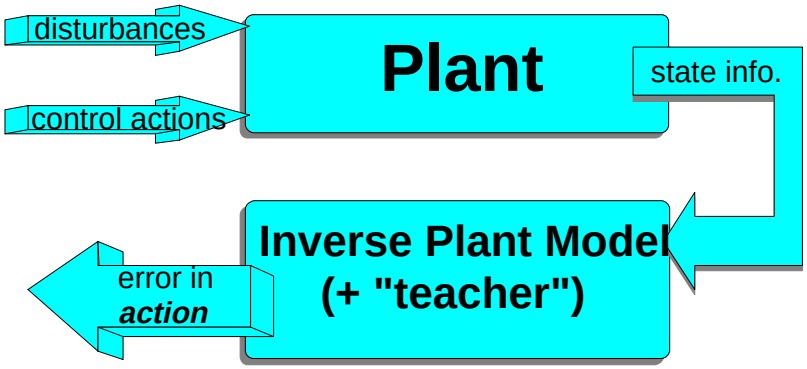


Figure B1.5.4. A supervised learning problem.

Now consider the *reinforcement learning* problem shown in figure B1.5.5 (Barto 1990). Here, no inverse plant model is available. However, a critic is available that indicates error in the state information from the plant. Because error is not directly provided in terms of control actions, the parameters of the controller cannot be directly adjusted by methods such as gradient descent.

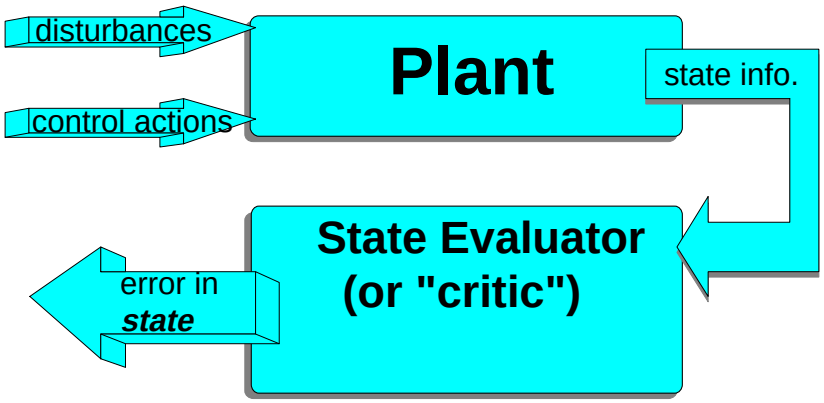


Figure B1.5.5. A reinforcement learning problem.

The remaining discussion will consider the control problem to operate as a *Markov decision problem*. That is, the control problem operates in discrete time steps, the plant is always in one of a finite number of discrete states, and a finite, discrete number of control actions are available. At each time step, the control action alters the probability of moving the plant from the current state to any other state. Note that deterministic environments are a specific case. Although this discussion will limit itself to discrete problems, most of the points made can be related directly to continuous problems.

A characteristic of many reinforcement learning problems is that one may need to consider a sequence of control actions and their results to determine how to improve the controller. One can examine the

implications of this by associating a *reward* or *cost* with each control action. The error in state in figure B1.5.5 can be thought of as a cost. One can consider the long-term effects of an action formally as the *expected, infinite-horizon discounted cost*:

$$\sum_{t=0}^{t=\infty} \lambda^t c_t$$

where $0 \leq \lambda \leq 1$ is the *discount parameter*, and c_t is the cost of the action taken at time t.

To describe a strategy for picking actions, consider the following approach: for each action u associated with a state i, assign a value $Q(i, u)$. A 'greedy' strategy is to select the action associated with the best Q at every time step. Therefore, an optimum setting for the Q-values is one in which a 'greedy' strategy leads to the minimum expected, infinite-horizon discounted cost. *Q-learning* is a method that yields optimal Q-values in restricted situations. Consider beginning with random settings for each Q-value, and updating each Q-value on-line as follows:

$$Q_{t+1}(i, u_t) = Q_t(i, u)(1 - \alpha) + \alpha \left[c_i(u_t) + \lambda \min Q(j, u_{t+1}) \right]$$

where $\min Q(j, u_{t+1})$ is the minimum Q available in state j, which is the state arrived in after action u_t is taken in state i (Barto *et al* 1991, Watkins 1989). The parameter α is a learning rate parameter that is typically set to a small value between zero and one. Arguments based on *dynamic programming* and *Bellman optimality* show that if each state–action pair is tried an infinite number of times, this procedure results in optimal Q-values. Certainly, it is impractical to try every state–action pair an infinite number of times. With finite exploration, Q-values can often be arrived at that are approximately optimal. Regardless of the method employed to update a strategy in a reinforcement learning problem, this exploration–exploitation dilemma always exists.

Another difficulty in the Q-value approach is that it requires storage of a separate Q-value for each state–action pair. In a more practical approach, one could store a Q-value for a group of state–action pairs that share the same characteristics. However, it is not clear how state–action pairs should be grouped. In many ways, the LCS can be thought of as a GA-based technique for grouping state–action pairs.

B1.5.2.3 Learning classifier system introduction

Consider the following method for representing a state–action pair in a reinforcement learning problem: encode a state in binary, and couple it to an action, which is also encoded in binary. In other words, the string

```
0 1 1 0 / 0 1 0
```

represents one of 16 states and one of eight actions. This string can also be seen as a rule that says 'IF in state `0 1 1 0`, THEN take action `0 1 0`'. In an LCS, such a rule is called a classifier. One can easily associate a Q-value, or other performance measures, with any given classifier.

Now consider generalizing over actions by introducing a 'don't care' character (#) into the state portion of a classifier. In other words, the string

```
# 1 1 # / 0 1 0
```

is a rule that says 'IF in state `0 1 1 0` OR state `0 1 1 1` OR state `1 1 1 0` OR state `1 1 1 1`, THEN take action `0 1 0`'. The introduction of this generality allows an LCS to represent clusters of states and associated actions. By using the genetic algorithm to search for such strings, one can search for ways of clustering states together, such that they can be assigned joint performance statistics, such as Q values.

Note, however, that Q-learning is not the most common method of credit assignment in LCSs. The most common method is called the *bucket brigade algorithm* for updating a classifier performance statistic called *strength*. Details of the bucket brigade algorithm will be introduced later in this section.

The structure of a typical LCS is shown in figure B1.5.6. This is what is known as a *stimulus–response* LCS, since no internal messages are used as memory. Details of internal message posting in LCSs will be discussed later. In this system, detectors encode state information from an environment into binary messages, which are matched against a list of rules called classifiers. The classifiers used are of the form

IF (condition) THEN (action).

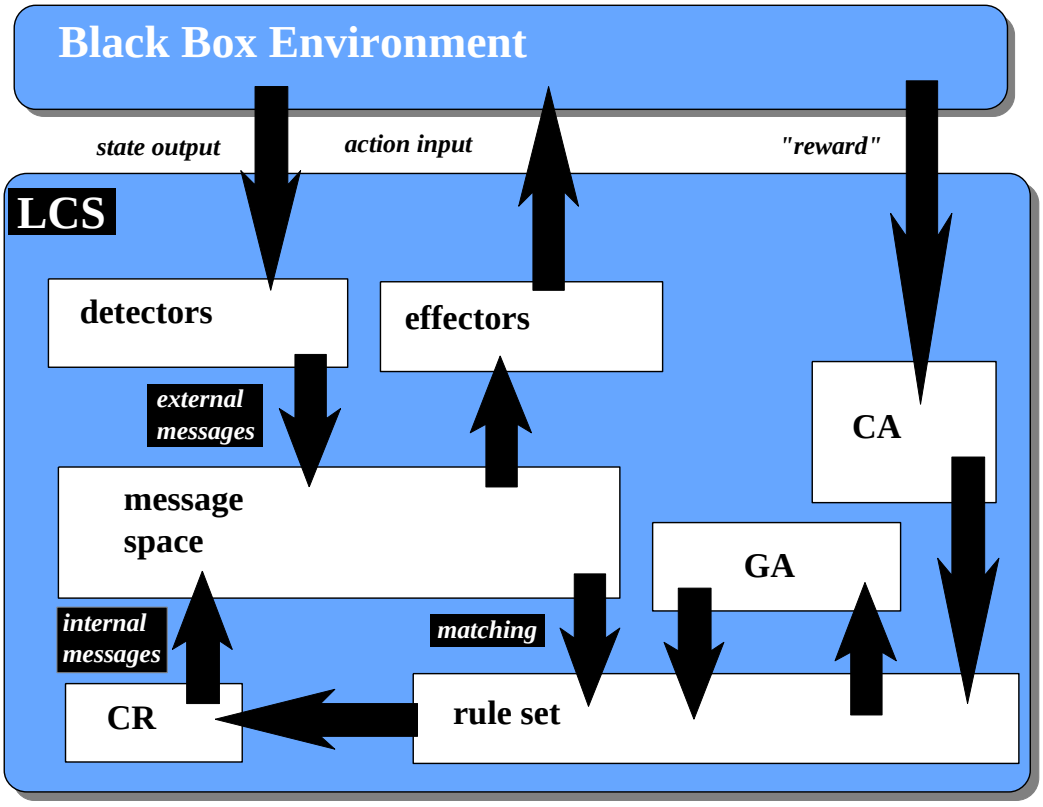


Figure B1.5.6. The structure of an LCS.

The operational cycle of this LCS is:

(i) Detectors post environmental messages on the message list.
(ii) All classifiers are matched against all messages on the message list.
(iii) Fully matched classifiers are selected to act.
(iv) A conflict resolution (CR) mechanism narrows the list of active classifiers to eliminate contradictory actions.
(v) The message list is cleared.
(vi) The CR-selected classifiers post their messages.
(vii) Effectors read the messages from the list, and take appropriate actions in the environment.
(viii) If a reward (or cost) signal is received, it is used by a credit allocation (CA) system to update parameters associated with the individual classifiers (such as the traditional strength measure, Q-like values, or other measures (Booker 1989, Smith 1991)).

B1.5.2.4 'Michigan' and 'Pitt' style learning classifier systems

There are two methods of using the genetic algorithm in LCSs. One is for each genetic algorithm population member to represent an entire set of rules for the problem at hand. This type of LCS is typified by Smith's LS-1 (Smith 1980), which was developed at the University of Pittsburgh. Often, this type of LCS is called the 'Pitt' approach. Another approach is for each genetic algorithm population member to represent a single rule. This type of LCS is typified by the CS-1 of Holland and Reitman (1978), which was developed at the University of Michigan, and is often called the 'Michigan' approach.

In the 'Pitt' approach, crossover and other operators are often employed that change the number of rules in any given population member. The 'Pitt' approach has the advantage of evaluating a complete solution within each genetic algorithm individual. Therefore, the genetic algorithm can converge to a homogeneous population, as in an optimization problem, with the best individual located by the genetic algorithm search acting as the solution. The disadvantage is that each genetic algorithm population member

must be completely evaluated as a rule set. This entails a large computational expense, and may preclude on-line learning in many situations.

In the 'Michigan' approach, one need only evaluate a single rule set, that comprised by the entire population. However, one cannot use the usual genetic algorithm procedures that will converge to a homogeneous population, since one rule is not likely to solve the entire problem. Therefore, one must *coevolve* a set of cooperative rules that jointly solve the problem. This requires a genetic algorithm procedure that yields a diverse population at steady state, in a fashion that is similar to *sharing* (Deb and Goldberg 1989, Goldberg and Richardson 1987), or other multimodal genetic algorithm procedures. In some cases simply dividing reward between similar classifiers that fire can yield sharing-like effects (Horn *et al* 1994). C6.1.2

B1.5.2.5 The bucket brigade algorithm (implicit form)

As was noted earlier, the bucket brigade algorithm is the most common form of credit allocation for LCSs. In the bucket brigade, each classifier has a strength, S, which plays a role analogous to a Q-value. The bucket brigade operates as follows:

(i) Classifier A is selected to act at time t.
(ii) Reward r_t is assigned in response to this action.
(iii) Classifier B is selected to act at time $t+1$.
(iv) The strength of classifier A is updated as follows:

$$S_{\mathrm{A}}^{t+1} = S_{\mathrm{A}}^{t}(1-\alpha) + \alpha\,[r_t + (\lambda S_{\mathrm{B}})]\,.$$

(v) The algorithm repeats.

Note that this is the *implicit* form of the bucket brigade, first introduced by Wilson (Goldberg 1989, Wilson 1985).

Note that this algorithm is essentially equivalent to Q-learning, but with one important difference. In this case, classifier A's strength is updated with the strength of the classifier that actually acts (classifier B). In Q-learning, the Q-value for the rule at time t is updated with the best Q-valued rule that matches the state at time $t+1$, whether that rule is selected to act at time $t+1$ or not. This difference is key to the convergence properties associated with Q-learning. However, it is interesting to note that recent empirical studies have indicated that the bucket brigade (and similar procedures) may be superior to Q-learning in some situations (Rummery and Niranjan 1994, Twardowski 1993).

A wide variety of variations of the bucket brigade exits. Some include a variety of *taxes*, which degrade strength based on the number of times a classifier has matched and fired and the number of generations since the classifier's creation. or other features. Some variations include a variety of methods for using classifier strength in conflict resolution through strength-based *bidding procedures* (Holland *et al* 1986). However, how these techniques fit into the broader context of machine learning, through similar algorithms such as Q-learning, remains a topic of research.

In many LCSs, strength is used as fitness in the genetic algorithm. However, a promising recent study indicates that other measures of classifier utility may be more effective (Wilson 1995).

B1.5.2.6 Internal messages

The LCS discussed to this point has operated entirely in stimulus–response mode. That is, it possesses no internal memory that influences which rule fires. In a more advanced form of the LCS, the action sections of the rule are internal messages that are posted on the message list. Classifiers have a condition that matches environmental messages (those which are posted by the environment) and a condition that matches internal messages (those posted by other classifiers). Some internal messages will cause effectors to fire (causing actions in the environment), and others simply act as internal memory for the LCS.

The operational cycle of a LCS with internal memory is as follows:

(i) Detectors post environmental messages on the message list.
(ii) All classifiers are matched against all messages on the message list.
(iii) Fully matched classifiers are selected to act.
(iv) A conflict resolution (CR) mechanism narrows the list of active classifiers to eliminate contradictory actions, and to cope with restrictions on the number of messages that can be posted.
(v) The message list is cleared.

(vi) The CR-selected classifiers post their messages.

(vii) Effectors read the messages from the list, and take appropriate actions in the environment.

(viii) If a reward (or cost) signal is received, it updates parameters associated with the individual classifiers.

In LCSs with internal messages, the bucket brigade can be used in its original, explicit form. In this form, the next rule that acts is 'linked' to the previous rule through an internal message. Otherwise, the mechanics are similar to those noted above. Once classifiers are linked by internal messages, they can form *rule chains* that express complex sequences of actions.

B1.5.2.7 Parasites

The possibility of rule chains introduced by internal messages, and by 'payback ' credit allocation schemes such as the bucket brigade or Q-learning, also introduces the possibility of rule *parasites*. Simply stated, parasites are rules that obtain fitness through their participation in a rule chain or a sequence of LCS actions, but serve no useful purpose in the problem environment. In some cases, parasite rules can prosper, while actually degrading overall system performance.

A simple example of parasite rules in LCSs is given by Smith (1994). In this study, a simple problem is constructed where the only performance objective is to exploit internal messages as internal memory. Although fairly effective rule sets were evolved in this problem, parasites evolved that exploited the bucket brigade, and the existing rule chains, but that were incorrect for overall system performance. This study speculates that such parasites may be an inevitable consequence in systems that use temporal credit assignment (such as the bucket brigade) and evolve internal memory processing.

B1.5.2.8 Variations of the learning classification system

As was stated earlier, this article only outlines the basic details of the LCS concept. It is important to note that many variations of the LCS exist. These include:

- *Variations in representation and matching procedures.* The {1, 0, #} representation is by no means defining to the LCS approach. For instance, Valenzuela-Rendón (1991) has experimented with a *fuzzy representation* of classifier conditions, actions, and internal messages. Higher-cardinality alphabets are also possible. Other variations include simple changes in the procedures that match classifiers to messages. For instance, sometimes partial matches between messages and classifier conditions are allowed (Booker 1982, 1985). In other systems, classifiers have multiple environmental or internal message conditions. In some suggested variations, multiple internal messages are allowed on the message list at the same time. D2
- *Variations in credit assignment.* As was noted above, a variety of credit assignment schemes can be used in LCSs. The examination of such schemes is the subject of much broader research in the reinforcement learning literature. Alternate schemes for the LCS prominently include *epochal* techniques, where the history of reward (or cost) signals is recorded for some period of time, and classifiers that act during the epoch are updated simultaneously.
- *Variations in discovery operators.* In addition to various versions of the genetic algorithm, LCSs often employ other discovery operators. The most common nongenetic discovery operators are those which create new rules to match messages for which no current rules exist. Such operators are often called *create*, *covering*, or *guessing* operators (Wilson 1985). Other covering operators are suggested that create new rules that suggest actions not accessible in the current rule set (Riolo 1986, Robertson 1988).

B1.5.2.9 Final comments

As was stated in section B1.5.2.1, the LCS remains a concept, more than a specific algorithm. Therefore, some of the details discussed here are necessarily sketchy. However, recent research on the LCS is promising. For a particularly clear examination of a simplified LCS, see a recent article by Wilson (1994). This article also recommends clear avenues for LCS research and development. Interesting LCS *applications* are also appearing in the literature (Smith and Dike 1995). G3.3

Given the robust character of evolutionary computation algorithms, the machine learning techniques suggested by the LCS concept indicate a powerful avenue of future evolutionary computation application.

B1.5.3 Hybrid methods

Zbigniew Michalewicz

Abstract

The concept of a hybrid evolutionary system is introduced here. Included are references to other sections in this handbook in which such hybrid systems are discussed in more detail.

There is some experimental evidence (Davis 1991, Michalewicz 1993) that the enhancement of evolutionary methods by some additional (problem-specific) heuristics, domain knowledge, or existing algorithms can result in a system with outstanding performance. Such enhanced systems are often referred to as *hybrid* evolutionary systems.

Several researchers have recognized the potential of such hybridization of evolutionary systems. Davis (1991, p 56) wrote:

> When I talk to the user, I explain that my plan is to hybridize the genetic algorithm technique and the current algorithm by employing the following three principles:
>
> - *Use the Current Encoding.* Use the current algorithm's encoding technique in the hybrid algorithm.
> - *Hybridize Where Possible.* Incorporate the positive features of the current algorithm in the hybrid algorithm.
> - *Adapt the Genetic Operators.* Create crossover and mutation operators for the new type of encoding by analogy with bit string crossover and mutation operators. Incorporate domain-based heuristics as operators as well.
>
> [...] I use the term *hybrid genetic algorithm* for algorithms created by applying these three principles.

The above three principles emerged as a result of countless experiments of many researchers, who tried to 'tune' their evolutionary algorithms to some problem at hand, that is, to create 'the best' algorithm for a particular class of problems. For example, during the last 15 years, various application-specific variations of evolutionary algorithms have been reported (Michalewicz 1996); these variations included variable-length strings (including strings whose elements were *if–then–else* rules), richer structures than binary strings, and experiments with modified genetic operators to meet the needs of particular applications. Some researchers (e.g. Grefenstette 1987) experimented with incorporating problem-specific knowledge into the initialization routine of an evolutionary system; if a (fast) heuristic algorithm provides individuals of the initial population for an evolutionary system, such a hybrid evolutionary system is guaranteed to do no worse than the heuristic algorithm which was used for the initialization.

Usually there exist several (better or worse) heuristic algorithms for a given problem. Apart from incorporating them for the purpose of initialization, some of these algorithms transform one solution into another by imposing a change in the solution's encoding (e.g. 2-opt step for the *traveling salesman problem*). One can incorporate such transformations into the operator set of evolutionary system, which usually is a very useful addition (see Chapter D3). G9.5 D3

Note also (see Sections C1.1 and C3.1) that there is a strong relationship between encodings of individuals in the population and operators, hence the operators of any evolutionary system must be chosen carefully in accordance with the selected representation of individuals. This is a responsibility of the developer of the system; again, we would cite Davis (1991, p 58): C1.1, C3.1

> Crossover operators, viewed in the abstract are operators that combine subparts of two parent chromosomes to produce new children. The adopted encoding technique should support operators of this type, but it is up to you to combine your understanding of the problem, the encoding technique, and the function of crossover in order to figure out what those operators will be. [...]
>
> The situation is similar for mutation operators. We have decided to use an encoding technique that is tailored to the problem domain; the creators of the current algorithm have done this tailoring for us. Viewed in the abstract, a mutation operator is an operator that introduces variations into the chromosome. [...] these variations can be global or local, but they are critical to keeping the genetic pot boiling. You will have to combine your knowledge of the problem,

the encoding technique, and the function of mutation in a genetic algorithm to develop one or more mutation operators for the problem domain.

Very often hybridization techniques make use of local search operators, which are considered as 'intelligent mutations'. For example, the best evolutionary algorithms for the traveling salesman problem use 2-opt or 3-opt procedures to improve the individuals in the population (see e.g. Mühlenbein *et al* 1988). It is not unusual to incorporate gradient-based (or hill-climbing) methods as ways for a local improvement of individuals. It is also not uncommon to combine *simulated annealing* techniques with some evolutionary algorithms (Adler 1993). D3.5

The class of hybrid evolutionary algorithms described so far consists of systems which extend evolutionary paradigm by incorporating additional features (local search, problem-specific representations and operators, and the like). This class also includes also so-called morphogenic evolutionary techniques (Angeline 1995), which include mappings (development functions) between representations that evolve (i.e. evolved representations) and representations which constitutes the input for the evaluation function (i.e. evaluated representations). However, there is another class of evolutionary hybrid methods, where the evolutionary algorithm acts as a separate component of a larger system. This is often the case for various scheduling systems, where the evolutionary algorithm is just responsible for ordering particular items (see, for example, Section F1.5). This is also the case for fuzzy systems, where the evolutionary algorithms may control the membership function (see Chapter D2), or of neural systems, where evolutionary algorithms may optimize the topology or weights of the network (see Chapter D1). F1.5 D2 D1

In this handbook there are several articles which refer (in a more or less explicit way) to the above classes of hybridization. In particular, Chapter C1 describes various representations, Chapter C3 appropriate operators for these representations, and Chapter D3 hybridizations of evolutionary methods with other optimization methods, whereas Chapters D1 and D2 provide an overview of neural–evolutionary and fuzzy–evolutionary systems, respectively. Also, many articles in Part G (Evolutionary Computation in Practice) describe evolutionary systems with hybrid components: it is apparent that hybridization techniques have generated a number of successful optimization algorithms. C1, C3 D3 D1, D2

References

Adler D 1993 Genetic algorithms and simulated annealing: a marriage proposal *Proc. IEEE Int. Conf. on Neural Networks* pp 1104–9

Angeline P J 1995 Morphogenic evolutionary computation: introduction, issues, and examples *Proc. 4th Ann. Conf. on Evolutionary Programming (San Diego, CA, March 1995)* ed J R McDonnell, R G Reynolds and D B Fogel (Cambridge, MA: MIT Press) pp 387–401

——1996 Two self-adaptive crossover operators for genetic programming *Advances in Genetic Programming 2* ed P J Angeline and K E Kinnear Jr (Cambridge, MA: MIT Press)

Angeline P J and Pollack J B 1993 Competitive environments evolve better solutions for complex tasks *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann)

Barto A G 1990 *Some Learning Tasks from a Control Perspective* COINS Technical Report 90-122, University of Massachusetts

Barto A G, Bradtke S J and Singh S P 1991 *Real-time Learning and Control using Asynchronous Dynamic Programming* COINS Technical Report 91-57, University of Massachusetts

Booker L B 1982 Intelligent behavior as an adaptation to the task environment *Dissertations Abstracts Int.* **43** 469B; University Microfilms 8214966

——1985 Improving the performance of genetic algorithms in classifier systems *Proc. Int. Conf. on Genetic Algorithms and Their Applications* pp 80–92

——1989 Triggered rule discovery in classifier systems *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, June 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 265–74

Cramer N L 1985 A representation of the adaptive generation of simple sequential programs *Proc. 1st Int. Conf. on Genetic Algorithms (Pittsburgh, PA, July 1985)* ed J J Grefenstette (Hillsdale, NJ: Erlbaum)

Davis L (ed) 1987 *Genetic Algorithms and Simulated Annealing* (Los Altos, CA: Morgan Kaufmann)

——1991 *Handbook of Genetic Algorithms* (New York: Van Nostrand Reinhold)

Deb K and Goldberg D E 1989 An investigation of niche and species formation in genetic function optimization *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, June 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 42–50

D'haeseleer P 1994 Context preserving crossover in genetic programming *1st IEEE Conf. on Evolutionary Computation (Orlando, FL, June 1994)* (Piscataway, NJ: IEEE)

Goldberg D E 1989 *Genetic Algorithms in Search, Optimization, and Machine Learning* (Reading, MA: Addison-Wesley)

Goldberg D E and Richardson J 1987 Genetic algorithms with sharing for multimodal function optimization *Proc. 2nd Int. Conf. on Genetic Algorithms (Cambridge, MA, 1987)* ed J J Grefenstette (Hillsdale, NJ: Erlbaum) pp 41–9

Grefenstette J J 1987 Incorporating problem specific knowledge into genetic algorithms *Genetic Algorithms and Simulated Annealing* ed L Davis (Los Altos, CA: Morgan Kaufmann) pp 42–60

Gruau F 1993 Genetic synthesis of modular neural networks *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann)

Holland J H 1975 *Adaptation in Natural and Artificial Systems* (Ann Arbor, MI: The University of Michigan Press)

Holland J H, Holyoak K J, Nisbett R E and Thagard P R 1986 *Induction: Processes of Inference, Learning, and Discovery* (Cambridge, MA: MIT Press)

Holland J H and Reitman J S 1978 Cognitive systems based on adaptive algorithms *Pattern Directed Inference Systems* ed D A Waterman and F Hayes-Roth (New York: Academic) pp 313–29

Horn J, Goldberg D E and Deb K 1994 Implicit niching in a learning classifier system: Nature's way *Evolutionary Comput.* **2** 37–66

Iba H, de Garis H and Sato T 1994 Genetic programming using a minimum description length principle *Advances in Genetic Programming* ed K E Kinnear Jr (Cambridge, MA: MIT Press)

Kauffman S A 1993 *The Origins of Order: Self-Organization and Selection in Evolution* (New York: Oxford University Press)

Kinnear K E Jr 1993 Generality and difficulty in genetic programming: evolving a sort *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July, 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann)

——1994 Alternatives in automatic function definition: a comparison of performance *Advances in Genetic Programming* ed K E Kinnear Jr (Cambridge, MA: MIT Press)

Koza J R 1989 Hierarchical genetic algorithms operating on populations of computer programs *Proc. 11th Int. Joint Conf. on Artificial Intelligence* (San Mateo, CA: Morgan Kaufmann)

——1990 *Genetic Programming: a Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems* Technical Report STAN-CS-90-1314, Computer Science Department, Stanford University

——1992 *Genetic Programming* (Cambridge, MA: MIT Press)

——1994 *Genetic Programming II* (Cambridge, MA: MIT Press)

Michalewicz Z 1993 A hierarchy of evolution programs: an experimental study *Evolutionary Comput.* **1** 51–76

——1996 *Genetic Algorithms + Data Structures = Evolution Programs* 3rd edn (New York: Springer)

Mühlenbein H, Gorges-Schleuter M and Krämer O 1988 Evolution algorithms in combinatorial optimization *Parallel Comput.* **7** 65–85

Nordin P 1994 A compiling genetic programming system that directly manipulates the machine code *Advances in Genetic Programming* ed K E Kinnear Jr (Cambridge, MA: MIT Press)

Perkis T 1994 Stack-based genetic programming *Proc. 1st IEEE Int. Conf. on Evolutionary Computation (Orlando, FL, June 1994)* (Piscataway, NJ: IEEE)

Reynolds C R 1994 Competition, coevolution and the game of tag *Artificial Life IV: Proc. 4th Int. Workshop on the Synthesis and Simulation of Living Systems* ed R A Brooks and P Maes (Cambridge, MA: MIT Press)

Riolo R L 1986 *CFS-C: a Package of Domain Independent Subroutines for Implementing Classifier Systems in Arbitrary User-defined Environments* University of Michigan, Logic of Computers Group, Technical Report

Robertson G G and Riolo R 1988 A tale of two classifier systems *Machine Learning* **3** 139–60

Rummery G A and Niranjan M 1994 *On-line Q-Learning using Connectionist Systems* Cambridge University Technical Report CUED/F-INFENG/TR 166

Sims K 1994 Evolving 3D morphology and behavior by competition *Artificial Life IV: Proc. 4th Int. Workshop on the Synthesis and Simulation of Living Systems* ed R A Brooks and P Maes (Cambridge, MA: MIT Press)

Smith R E 1991 *Default Hierarchy Formation and Memory Exploitation in Learning Classifier Systems* University of Alabama TCGA Report 91003; PhD Dissertation; University Microfilms 91-30 265

——1995 Memory exploitation in learning classifier systems *Evolutionary Comput.* **2** 199–220

Smith R E and Dike B A 1995 Learning novel fighter combat maneuver rules via genetic algorithms *Int. J. Expert Syst.* **8** 84–94

Smith S F 1980 A learning system based on genetic adaptive algorithms

Teller A 1996 Evolving programmers: the co-evolution of intelligent recombination operators *Advances in Genetic Programming 2* ed P J Angeline and K E Kinnear Jr (Cambridge, MA: MIT Press)

Twardowski K 1993 Credit assignment for pole balancing with learning classifier systems *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 238–45

Valenzuela-Rendón M 1991 The fuzzy classifier system: a classifier system for continuously varying variables *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, July 1991)* ed R Belew and L Booker (San Mateo, CA: Morgan Kaufmann) pp 346–53

Watkins J C H 1989 Learning with delayed rewards

Wilson S W 1985 Knowledge growth in an artificial animal *Proc. Int. Conf. on Genetic Algorithms and Their Applications* pp 16–23
——1994 ZCS: a zeroth level classifier system *Evolutionary Comput.* **2** 1–18
——1995 Classifier fitness based on accuracy *Evolutionary Comput.* **3** 149–76

Further reading

1. Koza J R 1992 *Genetic Programming* (Cambridge, MA: MIT Press)

   The first book on the subject. Contains full instructions on the possible details of carrying out genetic programming, as well as a complete explanation of genetic algorithms (on which genetic programming is based). Also contains 11 chapters showing applications of genetic programming to a wide variety of typical artificial intelligence, machine learning, and sometimes practical problems. Gives many examples of how to design a representation of a problem for genetic programming.

2. Koza J R 1994 *Genetic Programming II* (Cambridge, MA: MIT Press)

   A book principally about automatically defined functions (ADFs). Shows the applications of ADFs to a wide variety of problems. The problems shown in this volume are considerably more complex than those shown in *Genetic Programming*, and there is much less introductory material.

3. Kinnear K E Jr (ed) 1994 *Advances in Genetic Programming* (Cambridge, MA: MIT Press)

   Contains a short introduction to genetic programming, and 22 research papers on the theory, implementation, and application of genetic programming. The papers are typically longer than those in a technical conference and allow a deeper exploration of the topics involved. Shows a wide range of applications of genetic programming, as well as useful theory and practice in genetic programming implementation.

4. Forrest S (ed) 1993 *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* (San Mateo, CA: Morgan Kaufmann)

   Contains several interesting papers on genetic programming.

5. 1994 *1st IEEE Conf. on Evolutionary Computation (Orlando, FL, June 1994)* (Piscataway, NJ: IEEE)

   Contains many papers on genetic programming as well as a wide assortment of other EC-based papers.

6. Eshelman L J (ed) 1995 *Proc. 6th Int. Conf. on Genetic Algorithms (Pittsburgh, PA, July 1995)* (Cambridge, MA: MIT Press)

   Contains a considerable number of applications of genetic programming to increasingly diverse areas.

7. Angeline P J and Kinnear K E Jr (eds) 1996 *Advances in Genetic Programming II* (Cambridge, MA: MIT Press)

   A volume devoted exclusively to research papers on genetic programming, each longer and more in depth than those presented in most conference proceedings.

8. Kauffman S A 1993 *The Origins of Order: Self-Organization and Selection in Evolution* (New York: Oxford University Press)

   A tour-de-force of interesting ideas, many of them applicable to genetic programming as well as other branches of evolutionary computation.

9. ftp.io.com pub/genetic-programming

   An anonymous ftp site with considerable public domain information and implementations of genetic programming systems. This is a volunteer site, so its lifetime is unknown.

B2.1 Introduction

Nicholas J Radcliffe

Abstract

This section introduces the basic terminology of search, and reviews some of the key ideas underpinning its analysis. It begins with a discussion of search spaces, objective functions and the various possible goals of search. This leads on to a consideration of structure within search spaces, both in the form of a neighborhood structure and through the imposition of metrics over the space. Given such structure, various kinds of optima can be distinguished, including local, global and Pareto optima, all of which are defined. The main classes of operators typically used in search are then introduced, and the section closes with some remarks concerning the philosophy of search.

B2.1.1 Introduction

Evolutionary computation draws inspiration from natural evolving systems to build problem-solving algorithms. The range of problems amenable to solution by evolutionary methods includes optimization, constraint satisfaction, and covering, as well as more general forms of adaptation, but virtually all of the problems tackled are *search* problems. Our first definitions, therefore, concern search in general.

B2.1.2 Search

A *search space* $\mathcal{S}$ is a set of objects for potential consideration during search. Search spaces may be finite or infinite, continuous or discrete. For example, if the search problem at hand is that of finding a minimum of a function

$$F : \mathcal{X} \longrightarrow \mathcal{Y}$$

then the search space would typically be $\mathcal{X}$ or possibly some subset or superset of $\mathcal{X}$. Alternatively, if the problem is finding a computer program to test a number for primality, the search space might be chosen to be the set of LISP S-expressions using some given set of terminals and operators, perhaps limited to some depth of nesting (see Section B1.5.1). Notice that there is usually some choice in the definition of the search space, and that resolving this choice is part of the process of defining a well-posed search problem. Points in the search space $\mathcal{S}$ are usually referred to as *solutions* or *candidate solutions.*

B1.5.1

The goal of a search problem is usually to find one or more points in the search space having some specified property or properties. These properties are usually defined with respect to a function over the search space $\mathcal{S}$. This function, which generally takes the form

$$f : \mathcal{S} \longrightarrow \mathcal{R}$$

where $\mathcal{R}$ is most commonly the real numbers, $\mathbb{R}$, or some subset thereof, is known as the *objective function.*

When the search goal is optimization, the aim is to find one or more points in the search space which maximize f (in which case f is often known as a *utility* or *fitness* function, or occasionally a *figure of merit*), or which minimize f (in which case f is often known as a *cost function* or an *energy*). Of course, since $\max f = -\min(-f)$, maximization and minimization are equivalent.

When the goal is constraint satisfaction, f usually measures the degree to which a solution violates the constraints, and is called a *penalty function.* Here the goal is to find any zero of the penalty function.

Penalty functions are also commonly used to modify objective functions in optimization tasks in cases where there are constraints on the solution, though evolutionary methods also admit other approaches (see Section C5.2). C5.2

When the search task is a covering problem, the goal is usually to find a set of points in $\mathcal{S}$ which together minimize some function or satisfy some other properties. In this case, the objective function, which we will denote $\mathcal{O}$, usually takes the form

$$\mathcal{O} : \mathbb{P}(\mathcal{S}) \longrightarrow \mathcal{R}$$

where $\mathbb{P}(\mathcal{S})$ is the power set (i.e. set of all subsets) of $\mathcal{S}$, i.e. the objective function associates a value with a *set* of objects from the search space.

B2.1.3 Structure on the search space

B2.1.3.1 Global optima

Let us assume that the search problem is an optimization problem and, without loss of generality, that the goal is to minimize f. Then the *global optima* are precisely those points in $\mathcal{S}$ for which f is a minimum. The global optima are usually denoted with a star, so that

$$x^\star \in \mathcal{S}^\star \Longleftrightarrow f(x^\star) = \min_{x \in \mathcal{S}} f(x).$$

Notice that the global optima are always well defined provided that the objective function is well defined, and that they are independent of any structure defined on $\mathcal{S}$. For practical applications, it is often adequate to find instead a member of the *level set* defined by

$$L_{f^\star + \varepsilon} = \left\{ x \in \mathcal{S} \mid f(x) \leq f(x^\star) + \varepsilon \right\}$$

for some suitably chosen $\varepsilon > 0$.

B2.1.3.2 Neighborhoods and local optima

It is often useful to consider the search space $\mathcal{S}$ to be endowed with structure in the form of either a *connectivity* between its points or a *metric* (distance function) over it. In the case of a discrete space, the structure is usually associated with a *move operator*, M, which, starting from any point $x \in \mathcal{S}$, generates a *move* to a point $y \in \mathcal{S}$. The move operator considered is usually stochastic, so that different points y may be generated. The points which can be generated by a single application of the move operator from x are said to form the *neighborhood* of x, denoted $\mathrm{Nhd}(x)$. Those points that can be generated by up to k applications of M are sometimes known as the *k-neighborhood* of x, denoted $\mathrm{Nhd}(x, k)$. For example, if $\mathcal{S} = \mathbb{B}^4$ and the move operator M randomly changes a single bit in a solution, then the neighbors of 1010 are 0010, 1110, 1000, and 1011, while the 2-neighbors are 0110, 0000, 0011, 1100, 1111, and 1001. Assuming that M is symmetric (so that $y \in \mathrm{Nhd}(x, k) \Longleftrightarrow x \in \mathrm{Nhd}(y, k)$), a neighborhood structure automatically induces a connectivity on the space, together with a metric d given by

$$d(x, y) = \min \{k \in \mathbb{N} \mid y \in \mathrm{Nhd}(x, k)\}.$$

In the case of a continuous space, a metric may be similarly defined with respect to a move operator, but it is more commonly chosen to be one of the natural metrics on the space such as the Euclidean metric. In this case, rather than discrete neighborhoods, one talks of continuous *ε-neighborhoods.* The ε-neighborhood of a point $x \in \mathcal{S}$ is simply the set of points within distance ε of x with respect to the chosen metric d:

$$\mathrm{Nhd}(x, \varepsilon) = \{y \in \mathcal{S} \mid d(x, y) < \varepsilon\}.$$

Once a neighborhood structure has been established over $\mathcal{S}$, it becomes meaningful to talk of *local optima* (figure B2.1.2). In the case of a discrete space, a solution is said to be a local optimum if its objective function value is at least as low as that of each of its immediate neighbors. Thus the set of local optima $\mathcal{L} \subset \mathcal{S}$, defined with respect to the chosen neighborhood structure, is given by

$$\mathcal{L} = \{x \in \mathcal{S} \mid \forall y \in \mathrm{Nhd}(x) : f(x) \leq f(y)\}.$$

In the case of a continuous space, a local optimum is a point for which, for small enough ε, no member of its ε-neighborhood has a lower objective function value, so that

$$\mathcal{L} = \{x \in \mathcal{S} \mid \exists \varepsilon > 0 \; \forall y \in \mathrm{Nhd}(x, \varepsilon) : f(x) \leq f(y)\}.$$

Of course, all global optima are also local optima, though the term local optimum is often used loosely to refer to points which are locally but not globally optimal. A function is said to be *unimodal* if there is a unique global optimum and there are no nonglobal optima, and *multimodal* if there are multiple optima. (In fact, a search space that had multiple optima not separated by suboptimal solutions would also often be called unimodal, but such a definition depends on the structure imposed on the search space, so is avoided here.)

B2.1.3.3 Pareto optimality and multiple objectives

Another form of optimum arises when there are multiple objective functions, or, equivalently, when the objective function is vector valued. In such *multicriterion* or *multiobjective* problems, there is typically no solution that is 'better than' all others, but rather tradeoffs must be made between the various objective functions (Schaffer 1985, Fonseca and Fleming 1993). C4.5, F1.9

Suppose, without loss of generality, that the objective functions form the vector function

$$\boldsymbol{f} = (f_1, f_2, \ldots, f_n)$$

with

$$f_i : \mathcal{S} \longrightarrow \mathbb{R}$$

for each component f_i, and assume further that each function is to be minimized. A solution $x \in \mathcal{S}$ is now said to *dominate* another solution $y \in \mathcal{S}$ if it is no worse with respect to any component than y and is better with respect to at least one. Formally

$$x \text{ dominates } y \iff \begin{array}{ll} & \forall i \in \{1, 2, \ldots, n\} : f_i(x) \leq f_i(y) \\ \text{and} & \exists j \in \{1, 2, \ldots, n\} : f_j(x) < f_j(y). \end{array}$$

A solution is said to be *Pareto optimal* in $\mathcal{S}$ if it is not dominated by any other solution in $\mathcal{S}$, and the *Pareto-optimal set* or *Pareto-optimal front* is the set of such nondominated solutions, defined formally as

$$\mathcal{S}^\star = \{x \in \mathcal{S} \mid \nexists \, y \in \mathcal{S} : y \text{ dominates } x\}.$$

A typical Pareto-optimal front is shown in figure B2.1.1.

Multiobjective problems are usually formulated as covering problems, with the goal being to find the either the entire Pareto-optimal set, or a number of different points near it.

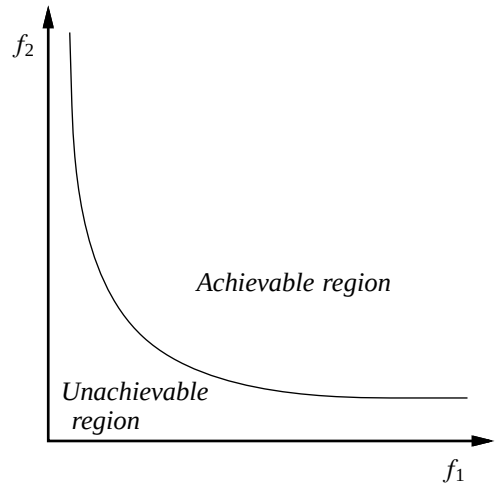


Figure B2.1.1. The *Pareto-optimal front* represents the best possible tradeoffs between two competing functions, both of which, in this case, are to be minimized. Each point on it represents the best (lowest) value for f_2 that can be achieved for a given value of f_1, and *vice versa*. All points in $\mathcal{S}$ have values for $\boldsymbol{f}$ on or above the front.

B2.1.4 Landscapes and energy surfaces

Intuitively, the reason that local optima are relevant to search is that in many circumstances it is easy to become 'trapped' in a local optimum, and for this to impede further progress. It is sometimes helpful to imagine the objective function as defining a *(fitness) landscape* (Wright 1932, Jones 1995) or an *(energy) surface* with points laid out to reflect the neighborhood structure discussed above. A solution then corresponds to a point on the landscape, and the height of that point represents its objective function value. Figure B2.1.2 can be considered to show the profile of a one-dimensional continuous landscape and an aerial view of a discrete landscape. If we again assume that the problem is a minimization problem, traditional gradient-based descent methods can be likened to releasing a marble from some point on the landscape: in all cases, the marble will come to rest at one of the minima of the landscape, but typically it will be a local rather than a global optimum.

B2.7

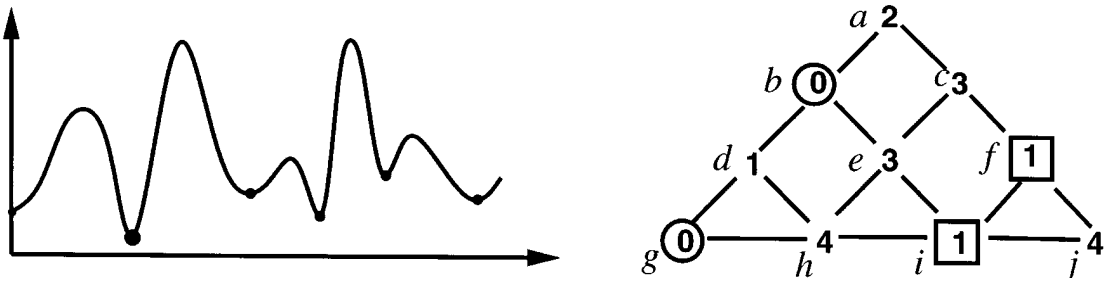


Figure B2.1.2. The left-hand graph shows the global minimum (large dot) of a continuous one-dimensional search space as well as the local optima. The right-hand graph shows the two global optima (in circles) and two local optima (squares) for a discrete space, where the numbers are function values and the letters are labels for later reference.

It is important to note, however, that the landscape structure is relevant to a search method only when the move operators used in that search method are strongly related to those which induce the neighborhood structure used to define the landscape. Referring again to the discrete landscape shown in figure B2.1.2, a move operator which moves only in the immediate neighborhood of any point will be unable to escape from the local optima marked with squares (f and i). However, a move operator which moves to points at exactly distance 2 from any starting point will see points d and f, but not i, as local optima.

While the intuition provided by the landscape analogy can be helpful in understanding search processes, it also carries some dangers. In particular, it is worth noting that the determination of the height of a point in the landscape involves computing the objective function. While in simple cases this may be a fast function evaluation, in real-world scenarios it is typically much more complex, perhaps involving running and taking measurements from a numerical simulation, or fitting a model to some parameters. In this sense, each point sampled from the landscape only comes into existence after the decision has been made to sample that solution. (There is thus no possibility of 'looking around' to spot points from a distance.) It is the evaluation of solutions that, in applications, usually takes the overwhelming proportion of the run time of an evolutionary algorithm.

B2.1.5 Representation

It is a particular characteristic of evolutionary computation that the search model in many cases defines moves not directly in the search space, but rather in an auxiliary space known as the *representation space*, I. Members of the *population*, which acts as the memory for an evolutionary algorithm, are members of this representation space, not of the search space. There are a number of reasons for this.

- Biologists make a strong distinction between the *genotype* that *specifies* an organism (its genetic code), and its observable characteristics, which together constitute its *phenotype.* For this reason, members of the search space $\mathcal{S}$ are often referred to as *phenotypes*, while members of the representation space I are called variously *genotypes, genomes,* or *chromosomes.* The term *individual* is also used, and usually refers to a member of the population—an element of the representation space.

- Many of the schools of evolutionary computation utilize standard move operators which are defined explicitly with respect to some representation. In this case, if the original search space is not the same as the representation space with respect to which these operators are defined, it is necessary to form a mapping between the two spaces.
- Certain of the evolutionary paradigms store within an individual not only information about the point in $\mathcal{S}$ represented, but also other information used to guide the search (Sections B1.3 and B1.4). Such auxiliary information usually takes the form of *strategy parameters*, which influence the moves made from the individual in question, and which are normally themselves subject to adaptation (Section C7.1). B1.3, B1.4 C7.1

Before an individual can be evaluated (with f), it must first be mapped to the search space. In deference to the evolutionary analogy, the process of transforming an individual genotype into a solution (phenotype) is often known as *morphogenesis*, and the function g that effects this is known as the *growth function*,

$$g : I \longrightarrow \mathcal{S}.$$

It should be clear that the choice of representation space and growth function is an important part of the strategy of using an evolutionary algorithm (Hart *et al* 1994).

The complexity of the growth function varies enormously between applications and between paradigms. In the simplest case (for example, real parameter optimization with an evolution strategy, or optimization of a Boolean function with a genetic algorithm) the growth function may be the identity mapping. In other cases the growth function may be stochastic, may involve repair procedures to produce a feasible (i.e. constraint-satisfying) solution from an infeasible one, or a legal (well-formed) solution from an illegal (ill-formed) individual. In more complex cases still, g may even involve a greedy construction of a solution from some starting information coded by the population member.

The representation space is sometimes larger than the search space (in which case either g is noninjective, and the representation is said to be *degenerate* (Radcliffe and Surry 1994), or some individuals are illegal), sometimes the same size (in which case, assuming that all solutions are represented, g is invertible), and occasionally smaller (in which case the global optima may not be represented).

The most common representations use a vector of values to represent a solution. The components of the vector are known as *genes*, again borrowing from the evolutionary analogy, and the values that a gene may take on are its *alleles*. For example, if n integers are used to represent solutions, and the ith may take values in the range $1–Z_i$, then the alleles for gene i are $\{1, 2, \ldots, Z_i\}$. If all combinations of assignments of allele values to genes are legal (i.e. represent members of the search space), the representation is said to be *orthogonal;* otherwise the representation is *nonorthogonal* (Radcliffe 1991).

B2.1.6 Operators

Evolutionary algorithms make use of two quite different kinds of operator. The first set are essentially independent of the details of the problem at hand, and even of the representation chosen for the problem. These are the operators for population maintenance—for example, *selection* and *replacement*, *migration*, and *deme management*. Move operators, on the other hand, are highly problem dependent, and fall into three main groups—*mutation operators*, *recombination operators*, and *local search* operators. C2, C6.3 C6.4

The main characteristic of *mutation* operators is that they operate on a single individual to produce a new individual. Most mutation operators, with typical parameter settings, have the characteristic that, for some suitably chosen metric on the space, they are relatively likely to generate offspring close to the parent solution, that is, within a small ε- or k-neighborhood. In some cases, the degree to which this is true is controlled by the strategy parameters for the individual undergoing mutation (Bäck and Schwefel 1993). Mutation operators are normally understood to serve two primary functions. The first function is as an exploratory move operator, used to generate new points in the space to test. The second is the maintenance of the 'gene pool'—the set of alleles available to recombination in the population. This is important because most recombination operators generate new solutions using only genetic 'material' available in the parent population. If the range of gene values in the population becomes small, the opportunity for recombination operators to perform useful search tends to diminish accordingly. C3.2

Recombination (or *crossover*) operators, the use of which is one of the more distinguishing features of many evolutionary algorithms, take two (or occasionally more) parents and produce from them one or C3.3

more offspring. These offspring normally take some characteristics from each of their parents, where a characteristic is usually, but not always, an explicit gene value.

Local search operators typically iteratively apply some form of unary move operator (often the mutation operator), accepting some or all improving moves. D3.2

B2.1.7 The process of search

The basic choice that all search algorithms have to make at each stage of the search is which point (or points) in the search space to sample next. At the start, assuming no prior knowledge, the choice is essentially random. After a while, as various points are sampled, and information is retained in whatever form of memory the search algorithm possesses, the choice becomes more informed. The likely effectiveness of the search is dependent on the strategy used to make choices on the basis of partial information about the search space and the degree to which the strategy is well matched to the problem at hand. In evolutionary algorithms, the population acts as the memory, and the choice of the next point in the search space to sample is determined by a combination of the individuals in the population and their objective function values, the representation used, and the move operators employed. It is important to note that the degree to which the move operators used are affected by any neighborhood structure imposed on the search space $\mathcal{S}$ depends on the interaction between the moves they effect in the representation space and the growth function g. In particular, local optima in the search space $\mathcal{S}$ under some given neighborhood structure may not be local optima in the representation space I for the chosen move operators. Conversely, there may be points in the representation space which do not correspond to obvious local optima in $\mathcal{S}$, but from which there is no (reasonably likely) move to a neighboring point (in I) with better objective function value under g. It is the latter points, rather than the former, which will tend to act as local traps for evolutionary algorithms.

B2.1.8 The philosophy of search

The ideal goals of search are:

- to find a global optimum (optimization)
- to find all global optima, or all non-dominated solutions (covering)
- to find a zero of the function (constraint satisfaction).

A more realistic goal, given some finite time and limited knowledge of the search space, is to make the maximum progress towards the appropriate goal—to find the best solution or solutions achievable in the time available. This point is reinforced by realizing that except in special cases, or with prior knowledge, it is not even possible to determine *whether* a given point is a global optimum without examining every point in the search space. Given the size and complexity of search spaces now regularly tackled, it is therefore rarely realistic to expect to find global optima, and to know that they are global optima.

There is an ongoing debate about whether or not evolutionary algorithms are properly classified as optimization methods *per se.* The striking elegance and efficiency of many of the results of natural evolution have certainly led many to argue that evolution is a process of optimization. Some of the more persuasive arguments for this position include the existence of organisms that exhibit components that are close to known mathematical optima to mechanical and other problems, and *convergent evolution*, whereby nearly identical designs evolve independently in nature.

Others argue that evolution, and evolutionary algorithms, are better described as *adaptative systems* (Holland 1975, De Jong 1992). Motivations for this view include the absence of guarantees of convergence of evolutionary search to solutions of any known quality, the changing environment in which natural evolution occurs, and arguments over 'fitness functions' in nature. Among others, Dawkins (1976) has argued particulary cogently that if evolution is to be understood as an optimization process at all, it is the propagation of DNA and the genes it contains that is maximized by evolution.

In a reference work such as this, it is perhaps best simply to note the different positions currently held within the field. Regardless of any eventual outcome, both the results of natural evolution and proven empirical work with evolutionary algorithms encourages the belief that, when applied to optimization problems, evolutionary methods are powerful global search methods.

References

Bäck T and Schwefel H-P 1993 An overview of evolutionary algorithms for parameter optimisation *Evolutionary Comput.* **1** 1–24

Dawkins R 1976 *The Selfish Gene* (Oxford: Oxford University Press)

De Jong K A 1992 Genetic algorithms are NOT function optimizers *Foundations of Genetic Algorithms 2* ed D Whitley (San Mateo, CA: Morgan Kaufmann) pp 2–18

Holland J H 1975 *Adaptation in Natural and Artificial Systems* (Ann Arbor, MI: University of Michigan Press)

Fonseca C M and Fleming P J 1993 Genetic algorithms for multiobjective optimization: formulation, discussion and generalization *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 416–23

Hart W, Kammeyer T and Belew R K 1994 The role of development in genetic algorithms *Foundations of Genetic Algorithms 3* ed D Whitley and M Vose (San Francisco, CA: Morgan Kaufmann) pp 315–32

Jones T C 1995 *Evolutionary Algorithms, Fitness Landscapes and Search* PhD Thesis, University of New Mexico

Radcliffe N J 1991 Forma analysis and random respectful recombination *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, July 1991)* ed R K Belew and L B Booker (San Mateo, CA: Morgan Kaufmann) pp 222–9

Radcliffe N J and Surry P D 1994 Fitness variance of formae and performance prediction *Foundations of Genetic Algorithms III* ed L D Whitley and M D Vose (San Mateo, CA: Morgan Kaufmann) pp 51–72

Schaffer J D 1985 Multiple objective optimization with vector evaluated genetic algorithms *Proc. 1st Int. Conf. on Genetic Algorithms (Pittsburgh, PA, July 1985* ed J J Grefenstette (Hillsdale, NJ: Erlbaum)

Wright S 1932 The roles of mutation, inbreeding, crossbreeding and selection in evolution *Proc. 6th Int. Congress on Genetics* vol 1, pp 256–366

B2.2 Stochastic processes

Günter Rudolph

Abstract

The purpose of this section is threefold. First, the notion of stochastic processes with particular emphasis on Markov chains and supermartingales is introduced and some general results are presented. Second, it is exemplarily shown how evolutionary algorithms (EAs) can be modeled by Markov chains. Third, some general sufficient conditions for EAs are derived to decide whether or not a particular EA converges to the global optimum.

B2.2.1 Stochastic processes in general

Historically, the term *stochastic process* has been reserved for families of random variables with some simple relationships between the variables (Doob 1967, p 47).

Definition B2.2.1. Let $(X_t : t \in T)$ be a family of random variables on a joint probability space $(\Omega, \mathcal{F}, \mathsf{P})$ with values in a set E of a measurable space $(E, \mathcal{A})$ and index set T. Then $(X_t : t \in T)$ is called a *stochastic process* with index set T. □

In general, there is no mathematical reason for restricting index set T to be a set of numerical values. In this section, however, the index set T is identical with $\mathbb{N}_0$ and the indices $t \in T$ will be interpreted as points of time.

Definition B2.2.2. A stochastic process $(X_t : t \in T)$ with index set $T = \mathbb{N}_0$ is called a *stochastic process with discrete time*. The sequence $X_0(\omega), X_1(\omega), \ldots$ is termed a *sample sequence* for each fixed $\omega \in \Omega$. The image space E of $(X_t : t \in T)$ is called the *state space* of the process. □

The next two subsections present some special cases of relationships that are important for the analysis of evolutionary algorithms.

B2.2.2 Markov chains

Stochastic processes possessing the *Markov property* (B2.2.1) below can be defined in a very general setting (Nummelin 1984). Although specializations to certain state spaces allow for considerable simplifications of the theory, the general case is presented first.

Definition B2.2.3. Let $(X_t : t \geq 0)$ be a stochastic process with discrete time on a probability space $(\Omega, \mathcal{F}, \mathsf{P})$ with values in E of a measurable space $(E, \mathcal{A})$. If for $0 < t_1 < t_2 < \cdots < t_k < t$ with some $k \in \mathbb{N}$ and $A \in \mathcal{A}$

$$\mathsf{P}\{X_t \in A \mid X_{t_1}, X_{t_2}, \ldots, X_{t_k}\} = \mathsf{P}\{X_t \in A \mid X_{t_k}\} \tag{B2.2.1}$$

almost surely then $(X_t : t \geq 0)$ is called a *Markov chain*. If $\mathsf{P}\{X_{t+k} \in A \mid X_{s+k}\} = \mathsf{P}\{X_t \in A \mid X_s\}$ for arbitrary $s, t, k \in \mathbb{N}_0$ with $s \leq t$, then the Markov chain is termed *homogeneous*, otherwise *inhomogeneous*. □ B2.3

Condition (B2.2.1) expresses the property that the behavior of the process after step t_k does not depend on states prior to step t_k provided that the state at step t_k is known.

Definition B2.2.4. Let $(X_t : t \geq 0)$ be a homogeneous Markov chain on a probability space $(\Omega, \mathcal{F}, \mathsf{P})$ with image space $(E, \mathcal{A})$. The map $\mathsf{K} : E \times \mathcal{A} \to [0, 1]$ is termed a *Markovian kernel* or a *transition probability function* for Markov chain $(X_t : t \geq 0)$ if $\mathsf{K}(\cdot, A)$ is measurable for any fixed set $A \in \mathcal{A}$ and $\mathsf{K}(x, \cdot)$ is a probability measure on $(E, \mathcal{A})$ for any fixed state $x \in E$. In particular, $\mathsf{K}(x_t, A) = \mathsf{P}\{X_{t+1} \in A \mid X_t = x_t\}$. □

The tth iteration of the Markovian kernel given by

$$\mathsf{K}^{(t)}(x, A) = \begin{cases} \mathsf{K}(x, A) & t = 1 \\ \int_E \mathsf{K}^{(t-1)}(y, A)\, \mathsf{K}(x, \mathrm{d}y) & t > 1 \end{cases}$$

describes the probability to transition to some set $A \subseteq E$ within t steps when starting from the state $x \in E$. Let $p(.)$ be the initial distribution over subsets A of $\mathcal{A}$. Then the probability that the Markov chain is in set A at step $t \geq 0$ is determined by

$$\mathsf{P}\{X_t \in A\} = \begin{cases} p(A) & t = 0 \\ \int_E \mathsf{K}^{(t)}(x, A)\, p(\mathrm{d}x) & t > 0 \end{cases}$$

where integration is with respect to an appropriate measure on $(E, \mathcal{A})$. For example, if $E = \mathbb{R}^n$ then integration is with respect to the Lebesgue measure. If E is finite then the counting measure is appropriate and the integrals reduce to sums. Then the Markovian kernel can be described by a finite number of transition probabilities $p_{xy} = \mathsf{K}(x, \{y\}) \geq 0$ that can be gathered in a square matrix $\mathbf{P} = (p_{xy})$ with $x, y \in E$. Since the state space is finite the states may be labeled uniquely by $1, 2, \ldots, c = \mathrm{card}(E)$ regardless of the true nature of the elements of E. To emphasize this labeling, the states from E are symbolized by i, j instead of x, y.

Obviously, the matrix $\mathbf{P}$ plays the role of the Markovian kernel in case of finite Markov chains. Therefore, each entry must be nonnegative and each row must add up to one in order to fulfill the requirements for a Markovian kernel. The tth iterate of the Markovian kernel corresponds to the tth power of matrix $\mathbf{P}$:

$$\mathbf{P}^t = \underbrace{\mathbf{P} \cdot \mathbf{P} \cdots \mathbf{P}}_{t \text{ times}}$$

for $t \geq 1$. Since matrix multiplication is associative the relation $\mathbf{P}^{t+s} = \mathbf{P}^t \cdot \mathbf{P}^s$ for $s, t \geq 0$ is valid and it is known as the *Chapman–Kolmogorov equation* for discrete Markov chains. By convention, $\mathbf{P}^0 = \mathbf{I}$, the unit matrix. The *initial distribution* $p_i := \mathsf{P}\{X_0 = i\}$ for $i \in E$ of the Markov chain can be gathered in a *row* vector $\boldsymbol{p} = (p_1, p_2, \ldots, p_c)$. Let $p_i^{(t)} := \mathsf{P}\{X_t = i\}$ denote the probability that the Markov chain is in state $i \in E$ at step $t \geq 0$ with $p_i^{(0)} := p_i$. Then

$$\boldsymbol{p}^{(t)} = \boldsymbol{p}^{(t-1)} \cdot \mathbf{P} = \boldsymbol{p}^{(0)} \cdot \mathbf{P}^t$$

for $t \geq 1$. Therefore, a homogeneous finite Markov chain is completely determined by the pair $(\boldsymbol{p}^{(0)}, \mathbf{P})$. Evidently, the limit behavior of the Markov chain depends on the iterates of the Markovian kernel and therefore on the structure of the transition matrix. For a classification some definitions are necessary (Seneta 1981, Minc 1988).

Definition B2.2.5. A square matrix $\mathbf{V} : c \times c$ is called a *permutation matrix* if each row and each column contain exactly one 1 and $c - 1$ zeros. A matrix $\mathbf{A}$ is said to be *cogredient* to a matrix $\mathbf{B}$ if there exists a permutation matrix $\mathbf{V}$ such that $\mathbf{A} = \mathbf{V}'\mathbf{B}\mathbf{V}$. A square matrix $\mathbf{A}$ is said to be *nonnegative* (*positive*), denoted $\mathbf{A} \geq \mathbf{0}$ ($> \mathbf{0}$), if $a_{ij} \geq 0$ (> 0) for each entry a_{ij} of $\mathbf{A}$. A nonnegative matrix is called *reducible* if it is cogredient to a matrix of the form

$$\begin{pmatrix} \mathbf{C} & \mathbf{0} \\ \mathbf{R} & \mathbf{T} \end{pmatrix}$$

where $\mathbf{C}$ and $\mathbf{T}$ are square matrices. Otherwise, the matrix is called *irreducible*. An irreducible matrix is called *primitive* if there exists a finite constant $k \in \mathbb{N}$ such that its kth power is positive. A nonnegative matrix is said to be *stochastic* if all its row sums are unity. A stochastic matrix with identical rows is termed *stable*. □

Note that the product of stochastic matrices is again a stochastic matrix and that every positive matrix is also primitive. Clearly, transition matrices are stochastic and they can be brought into some *normal form*:

Theorem B2.2.1 (Iosifescu 1980, p 95). Each transition matrix of a homogeneous finite Markov chain is cogredient to one of the following *normal forms*:

$$\mathbf{P}_1 = \begin{pmatrix} \mathbf{C}_1 & & & \\ & \mathbf{C}_2 & & \\ & & \ddots & \\ & & & \mathbf{C}_r \end{pmatrix} \quad \text{or} \quad \mathbf{P}_2 = \begin{pmatrix} \mathbf{C}_1 & & & & \\ & \mathbf{C}_2 & & & \\ & & \ddots & & \\ & & & \mathbf{C}_r & \\ \mathbf{R}_1 & \mathbf{R}_2 & \cdots & \mathbf{R}_r & \mathbf{T} \end{pmatrix}$$

where submatrices $\mathbf{C}_1, \ldots, \mathbf{C}_r$ with $r \geq 1$ are irreducible and at least one of the submatrices $\mathbf{R}_i$ is nonzero. □

To proceed the following terms have to be introduced:

Definition B2.2.6. Let $\mathbf{P}$ be the transition matrix of a homogeneous finite Markov chain. A distribution $\boldsymbol{p}$ on the states of the Markov chain is called a *stationary distribution* if $\boldsymbol{p}\mathbf{P} = \boldsymbol{p}$ and a *limit distribution* if the limit $\boldsymbol{p} = \boldsymbol{p}^{(0)} \lim_{t\to\infty} \mathbf{P}^t$ exists. □

Now some limit theorems may be stated:

Theorem B2.2.2 (Iosifescu 1980, p 123, Seneta 1981, p 119). Let $\mathbf{P}$ be a primitive stochastic matrix. Then $\mathbf{P}^t$ converges as $t \to \infty$ to a positive stable stochastic matrix $\mathbf{P}^\infty = \mathbf{1}'\boldsymbol{p}^{(\infty)}$, where the limit distribution $\boldsymbol{p}^{(\infty)} = \boldsymbol{p}^{(0)} \lim_{t\to\infty} \mathbf{P}^t = \boldsymbol{p}^{(0)}\mathbf{P}^\infty$ has nonzero entries and is unique regardless of the initial distribution. Moreover, the limit distribution is identical with the unique stationary distribution and is given by the solution $\boldsymbol{p}^{(\infty)}$ of the system of linear equations $\boldsymbol{p}^{(\infty)}\mathbf{P} = \boldsymbol{p}^{(\infty)}$, $\boldsymbol{p}^{(\infty)}\mathbf{1}' = \mathbf{1}$. □

Theorem B2.2.3 (Goodman 1988, pp 158–9). Let $\mathbf{P} : c \times c$ be a reducible stochastic matrix cogredient to the normal form

$$\mathbf{P} = \begin{pmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{R} & \mathbf{T} \end{pmatrix}$$

where $\mathbf{I}$ has rank $m < c$. Then the iterates of $\mathbf{P}$ converge to

$$\mathbf{P}^{(\infty)} = \lim_{t\to\infty} \mathbf{P}^t = \begin{pmatrix} \mathbf{I} & \mathbf{0} \\ \sum_{k=0}^{t-1} \mathbf{T}^k\mathbf{R} & \mathbf{T}^t \end{pmatrix} = \begin{pmatrix} \mathbf{I} & \mathbf{0} \\ (\mathbf{I}-\mathbf{T})^{-1}\mathbf{R} & \mathbf{0} \end{pmatrix}$$

and the limit distribution $\boldsymbol{p}^{(\infty)}$ satisfies $p_i^{(\infty)} = 0$ for $m < i \leq c$ and $\sum_{i=1}^m p_i^{(\infty)} = 1$ regardless of the initial distribution. □

A variation of theorem B2.2.3 is given below.

Theorem B2.2.4 (Iosifescu 1980, p 126, Seneta 1981, p 127). Let $\mathbf{P} : c \times c$ be a reducible stochastic matrix cogredient to the normal form

$$\mathbf{P} = \begin{pmatrix} \mathbf{C} & \mathbf{0} \\ \mathbf{R} & \mathbf{T} \end{pmatrix}$$

where $\mathbf{C} : m \times m$ is a primitive stochastic matrix and $\mathbf{R}, \mathbf{T} \neq \mathbf{0}$. Then

$$\mathbf{P}^\infty = \lim_{t\to\infty} \mathbf{P}^t = \lim_{t\to\infty} \begin{pmatrix} \mathbf{C}^t & \mathbf{0} \\ \sum_{k=0}^{t-1} \mathbf{T}^k\mathbf{R}\mathbf{C}^{t-k} & \mathbf{T}^t \end{pmatrix} = \begin{pmatrix} \mathbf{C}^\infty & \mathbf{0} \\ \mathbf{R}_\infty & \mathbf{0} \end{pmatrix}$$

is a stable stochastic matrix with $\mathbf{P}^\infty = \mathbf{1}'\boldsymbol{p}^\infty$, where $\boldsymbol{p}^{(\infty)} = \boldsymbol{p}^{(0)}\mathbf{P}^\infty$ is unique regardless of the initial distribution, and $\boldsymbol{p}^{(\infty)}$ satisfies $p_i^{(\infty)} > 0$ for $1 \leq i \leq m$ and $p_i^{(\infty)} = 0$ for $m < i \leq c$. □

In the literature some efforts have been devoted to the question of how fast the distribution $\boldsymbol{p}^{(t)}$ of the Markov chain approaches the limit distribution. It can be shown that $\|\boldsymbol{p}^{(t)} - \boldsymbol{p}^{(\infty)}\| = \mathrm{O}(t^a\beta^t)$ with $\beta \in (0, 1)$ and $a \geq 0$ for the transition matrices treated in theorems B2.2.2–B2.2.4 (see e.g. Isaacson and Madsen 1976, Iosifescu 1980, Rosenthal 1995). As for *global convergence* rates of evolutionary algorithms B2.3 with finite search spaces, however, it will suffice to know the rates at which the iterates of matrix $\mathbf{T}$ in theorems B2.2.3 and B2.2.4 approach the zero matrix. Since the limit distributions are approached with a

geometric rate it is clear that the rate for matrix $\mathbf{T}$ is geometric as well, but the bound for the latter may be smaller.

After having dealt with the limit behavior of some Markov chains the next question concerns the time that must be awaited on average to reach a specific set of states.

Definition B2.2.7. The random time $H_A = \min\{t \geq 0 : X_t \in A \subseteq E\}$ is called the *first hitting (entry, passage) time* of set $A \subseteq E$ while $L_A = \max\{t \geq 0 : X_t \in A \subseteq E\}$ is termed the *last exit time*. A non-empty set $A \subseteq E$ is called *transient* if $\mathsf{P}\{L_A < \infty \mid X_0 = x\} = 1$ for all $x \in A$. If $\mathsf{K}(x, A) = 1$ for all $x \in A$ then the set A is called *absorbing* (under Markovian kernel K). If A is absorbing then H_A is called the *absorption time*. □

Suppose that the state space E can be decomposed into two disjoint sets A and T where A is absorbing and T is transient. For finite state spaces this situation is reflected by theorems B2.2.3 and B2.2.4 in which the transient states are associated with matrix $\mathbf{T}$ whereas the absorbing set is associated with matrices $\mathbf{I}$ and $\mathbf{C}$, respectively.

Obviously, as $H_A = L_T + 1$, provided that $X_0 \in T$, it is sufficient to count the number of times that the Markov chain is in the transient set in order to obtain the absorption time to set A. Consequently, the expected absorption time is given by

$$\begin{aligned}\mathsf{E}[H_A \mid X_0 = x] &= \sum_{t=0}^{\infty} \mathsf{E}[1_T(X_t) \mid X_0 = x] = \sum_{t=0}^{\infty} (1 \cdot \mathsf{P}\{X_t \in T \mid X_0 = x\} + 0 \cdot \mathsf{P}\{X_t \notin T \mid X_0 = x\}) \\ &= \sum_{t=0}^{\infty} \mathsf{P}\{X_t \in T \mid X_0 = x\} = \sum_{t=0}^{\infty} \mathsf{K}^{(t)}(x, T)\end{aligned}$$

where $\mathsf{K}^{(0)}(x, T) = 1_T(x)$. In the finite case the Markovian kernel $\mathsf{K}(i, \{j\})$ for $i, j \in T$ is represented by matrix $\mathbf{T}$. Since $\sum_{t \geq 0} \mathbf{T}^t = (\mathbf{I} - \mathbf{T})^{-1}$ one obtains the following result.

Theorem B2.2.5 (Iosifescu 1980, p 104, Seneta 1981, p 122). If a_i denotes the random time until absorption from state $i \in E$ and $\boldsymbol{a} = (a_1, a_2, \ldots, a_c)$ then $\mathsf{E}[\boldsymbol{a}] = (\mathbf{I} - \mathbf{T})^{-1}\mathbf{1}'$. □

B2.2.3 Supermartingales

The next special case of stochastic processes considered here deals with those processes that have a relationship between random variable X_t and the conditional expectation of X_{t+1} (Neveu 1975).

Definition B2.2.8. Let $(\Omega, \mathcal{F}, \mathsf{P})$ be a probability space and $\mathcal{F}_0 \subseteq \mathcal{F}_1 \subseteq \ldots \mathcal{F}$ be an increasing family of sub-σ-algebras of $\mathcal{F}$ and $\mathcal{F}_\infty := \sigma\left(\bigcup_t \mathcal{F}_t\right) \subseteq \mathcal{F}$. A stochastic process (X_t) that is $\mathcal{F}_t$-measurable for each t is termed a *supermartingale* if

$$\mathsf{E}[\mid X_t \mid] < \infty \qquad \text{and} \qquad \mathsf{E}[X_{t+1} \mid \mathcal{F}_t] \leq X_t$$

for all $t \geq 0$. If $\mathsf{P}\{X_t \geq 0\} = 1$ for all $t \geq 0$ then the supermartingale is said to be *nonnegative*. □

Nonnegative supermartingales have the following remarkable property.

Theorem B2.2.6 (Neveu 1975, p 26). If $(X_t : t \geq 0)$ is a nonnegative supermartingale then $X_t \overset{\text{a.s.}}{\to} X < \infty$. □

Although nonnegative supermartingales do converge almost surely to a finite limit, nothing can be said about the limit itself unless additional conditions are imposed. For later purposes it is of interest to know under which conditions the limit is the constant zero. The proof of the following result is given by Rudolph (1994a).

Theorem B2.2.7. If $(X_t : t \geq 0)$ is a nonnegative supermartingale satisfying $\mathsf{E}[X_{t+1} \mid \mathcal{F}_t] \leq c_t X_t$ almost surely for all $t \geq 0$ with $c_t \geq 0$ and

$$\sum_{t=1}^{\infty} \left(\prod_{k=0}^{t-1} c_k \right) < \infty \tag{B2.2.2}$$

then $X_t \overset{\text{m}}{\to} 0$ and $X_t \overset{\text{c}}{\to} 0$ as $t \to \infty$. □ B2.3

Condition (B2.2.2) is fulfilled if for example $\limsup\{c_t : t \geq 0\} < 1$.

B2.2.4 Stochastic process models of evolutionary algorithms

Evolutionary algorithms (EAs) can be modeled almost always as homogeneous Markov chains. For this purpose one has to define an appropriate state space E and the probabilistic behavior of the evolutionary operators (variation and selection operators) must be expressed in terms of transition probabilities (i.e. the Markovian kernel) over this state space. The general technique to derive the Markovian kernel K rests on its property that it can be decomposed into $k < \infty$ mutually independent Markovian kernels $\mathsf{K}_1, \ldots, \mathsf{K}_k$—each of them describing an evolutionary operator—such that K is just their product kernel

$$\begin{aligned}\mathsf{K}(x, A) &= (\mathsf{K}_1\mathsf{K}_2 \cdots \mathsf{K}_k)(x, A) \\ &= \int_E \mathsf{K}_1(x_1, \mathrm{d}x_2) \int_E \mathsf{K}_2(x_2, \mathrm{d}x_3) \cdots \int_E \mathsf{K}_{k-2}(x_{k-2}, \mathrm{d}x_{k-1}) \int_E \mathsf{K}_{k-1}(x_{k-1}, \mathrm{d}x_k)\, \mathsf{K}_k(x_k, A)\end{aligned}$$

with $x_1 = x \in E$ and $A \subseteq E$. Evidently, for finite state spaces the Markovian kernels for the evolutionary operators are transition matrices and the product kernel is just the product of these matrices.

If I is the space representing admissible instances of an individual, then the most natural way to define the state space E of an evolutionary algorithm with μ individuals is given by $E = I^\mu$. Mostly this choice exhibits some redundancy, because the actual arrangement of the individuals in the population is seldom of importance. Especially for finite I with cardinality s the state space $E = I^\mu$ can be condensed to the smaller state space $E' = \{\boldsymbol{x} \in \mathbb{N}_0^s : \|\boldsymbol{x}\|_1 = \mu\}$. Here, each entry x_j of $\boldsymbol{x}$ represents the number of individuals of type $i_j \in I$, where the elements of I are uniquely labeled from 1 to $s < \infty$. This type of state space was often used to build an exact Markov model of an evolutionary algorithm for binary finite search spaces with *proportionate selection*, *bit-flipping mutation*, and *one-point crossover* (Davis 1991, Nix and Vose 1992, Davis and Principe 1993, and others). In order to obtain global convergence results *qualitative* Markov models of evolutionary algorithms are sufficient: see the articles by Fogel (1994), Rudolph (1994b), and Suzuki (1995) for qualitative Markovian models of EAs with finite search spaces. C2.2, C3.2.1, C3.3.1

The essence of the above-mentioned references is that EAs on binary search spaces—as they are commonly used—can be divided into two classes, provided that *bit-flipping mutation* is used: the transition matrix is primitive if the selection operator is nonelitist, while the matrix is reducible if the selection operator is elitist. For example, let $I = \mathbb{B}^\ell$ and $E = I^\mu$. Since each bit in $i \in E$ is mutated independently with some probability $p \in (0, 1)$, the transition probability m_{ij} to mutate population i to population $j \in E$ is $m_{ij} = p^{h(i,j)}\,(1-p)^{\mu\ell - h(i,j)} > 0$ where $h(i, j)$ is the Hamming distance between i and j. Consequently, the transition matrix for mutation $\mathbf{M} = (m_{ij})$ is positive. Let $\mathbf{C}$ be the stochastic matrix gathering the transition probabilities for some crossover operator and $\mathbf{S}$ the transition matrix for selection. It is easy to see that the product $\mathbf{CM}$ is positive. If there exists a positive probability to select exactly the same population as the one given prior to selection (which is true for proportionate stochastic, q-ary tournament and some other selection rules), then the main diagonal entries of matrix $\mathbf{S}$ are positive and the transition matrix of the entire EA is positive: $\mathbf{P} = \mathbf{CMS} > \mathbf{0}$. C3.2.1

Finally, consider a $(1 + 1)$ EA with search space $\mathbb{R}^n$. An individual $\boldsymbol{X}_t$ is mutated by adding a normally distributed random vector $\boldsymbol{Z}_t \sim N(\mathbf{0}, \sigma^2\,\mathbf{I})$ with $\sigma > 0$ where the sequence $(\boldsymbol{Z}_t : t \geq 0)$ is independent and identical distributed. The mutated point $\boldsymbol{Y}_t = \boldsymbol{X}_t + \boldsymbol{Z}_t$ is selected to serve as parent for the next generation if it is better than or equal to $\boldsymbol{X}_t$: $f(\boldsymbol{Y}_t) \leq f(\boldsymbol{X}_t)$ in the case of minimization.

To model this EA in a Markovian framework choose $E = I = \mathbb{R}^n$. Then the mutation kernel is given by

$$\mathsf{K}_\mathrm{m}(x, A) = \int_A f_Z(z - x)\,\mathrm{d}z$$

where $x \in E$, $A \subseteq E$, and f_Z is the probability density function of random vector $\boldsymbol{Z} \sim N(\mathbf{0}, \sigma^2\,\mathbf{I})$. Let $B(x) = \{y \in E : f(y) \leq f(x)\}$ be the set of admissible solutions with a quality better than or equal to the quality of solution $x \in E$. Since the selection kernel depends on the previous state $x \in E$ this state is attached to K_s as an additional parameter. Then the selection kernel is given by

$$\mathsf{K}_\mathrm{s}(y, A; x) = 1_{B(x)}(y) \cdot 1_A(y) + 1_{B^\mathrm{c}(x)}(y) \cdot 1_A(x) \tag{B2.2.3}$$

and may be interpreted as follows. If state $y \in E$ is better than or equal to state x (i.e. $y \in B(x)$) and also in set A, then y transitions to set A, and more precisely to set $A \cap B(x)$, with probability one. If y is worse than x (i.e. $y \in B^\mathrm{c}(x)$) then y is not accepted. Rather, y will transition to the old state x with

probability one. But if x is in set A then y will transition to $x \in A$ with probability one. All other cases have probability zero. Evidently, the selection kernel is purely deterministic here. Putting all this together the product kernel of mutation and selection is

$$\mathsf{K}(x, A) = \int_E \mathsf{K}_\mathrm{m}(x, \mathrm{d}y) \cdot \mathsf{K}_\mathrm{s}(y, A; x) = \int_E \mathsf{K}_\mathrm{m}(x, \mathrm{d}y) \cdot 1_{A \cap B(x)}(y) + 1_A(x) \cdot \int_E \mathsf{K}_\mathrm{m}(x, \mathrm{d}y) \cdot 1_{B^\mathrm{c}(x)}(y)$$
$$= \int_{A \cap B(x)} \mathsf{K}_\mathrm{m}(x, \mathrm{d}y) + 1_A(x) \cdot \int_{B^\mathrm{c}(x)} \mathsf{K}_\mathrm{m}(x, \mathrm{d}y) = \mathsf{K}_\mathrm{m}(x, A \cap B(x)) + 1_A(x) \cdot \mathsf{K}_\mathrm{m}(x, B^\mathrm{c}(x)).$$

There are two important observations: First, the above formula remains valid for arbitrary state space E, only the integration must be done with respect to the appropriate measure. Second, kernel K_m may be interpreted as a Markovian kernel describing all evolutionary operators that modify state $x \in E$ to generate a new trial point in E.

As a consequence, the structure of kernel K remains valid for population-based EAs with arbitrary search spaces and (a special version of) elitist selection. To see this let $E = I^\mu$ with arbitrary I and recall the definition of *map* $b : E \to I$ that extracts the best individual from a population. Then the set of states B2.3 better than or equal to state x can be redefined via $B(x) = \{y \in E : f(b(y)) \leq f(b(x))\}$.

What happens with the selection kernel? If $y \in E$ is in $B(x) \cap A$ the population transitions to A. If $y \notin B(x)$ then the best individual of population y is worse than the best individual of population x. If the entire population is rejected the kernel is identical to (B2.2.3). However under usual elitist selection the best individual $b(x)$ is reinserted—somehow—into population y yielding $y' = e(x, y) \in B(x)$. Here the map $e : E \times E \to E$ encapsulates the method to reinsert the best individual $b(x)$ into y. Consequently, the selection kernel becomes

$$\mathsf{K}_\mathrm{s}(y, A; x) = 1_{B(x) \cap A}(y) + 1_{B^\mathrm{c}(x)}(y) \cdot 1_A(x) \cdot 1_A(e(x, y))$$

leading to

$$\mathsf{K}(x, A) = \mathsf{K}_\mathrm{m}(x, B(x) \cap A) + 1_A(x) \cdot \int_{B^\mathrm{c}(x)} \mathsf{K}_\mathrm{m}(x, \mathrm{d}y) \cdot 1_A(e(x, y)). \tag{B2.2.4}$$

The integral in (B2.2.4) is unpleasant, but in the next section it will be investigated whether some EA is able to converge in some sense to a specific set A_ϵ that is related to the globally optimal solutions of an optimization problem. Restricted to this set A_ϵ the Markovian kernel shrinks to a very simple expression.

B2.2.5 Convergence conditions for evolutionary algorithms

Let $A_\epsilon = \{x \in E : f(b(y)) - f^* \leq \epsilon\}$ for some $\epsilon > 0$ where f^* is the global minimum of the objective function. The main convergence condition is given below (the proof can be found in the article by Rudolph (1996)).

Theorem B2.2.8. An evolutionary algorithm, whose Markovian kernel satisfies the conditions $\mathsf{K}(x, A_\epsilon) \geq \delta > 0$ for all $x \in A_\epsilon^\mathrm{c} = E \setminus A_\epsilon$ and $\mathsf{K}(x, A_\epsilon) = 1$ for $x \in A_\epsilon$ will converge completely to the global minimum of a real-valued function f defined on an arbitrary search space, provided that f is bounded from below. □

But which evolutionary algorithms possess a Markovian kernel that satisfies the preconditions of theorem B2.2.8? To answer the question consider EAs whose Markovian kernel is represented by (B2.2.4). If $A_\epsilon \subset B(x)$ then $x \notin A_\epsilon$, $A_\epsilon \cap B(x) = A_\epsilon$ and $\mathsf{K}(x, A_\epsilon) = \mathsf{K}_\mathrm{m}(x, A_\epsilon)$. If $B(x) \subseteq A_\epsilon$ then $x \in A_\epsilon$, $A_\epsilon \cap B(x) = B(x)$ and

$$\mathsf{K}(x, A_\epsilon) = \mathsf{K}_\mathrm{m}(x, B(x)) + \int_{B^\mathrm{c}(x)} \mathsf{K}_\mathrm{m}(x, \mathrm{d}y) \cdot 1_{A_\epsilon}(e(x, y)) = \mathsf{K}_\mathrm{m}(x, B(x)) + \mathsf{K}_\mathrm{m}(x, B^\mathrm{c}(x)) = 1$$

since $e(x, y) \in B(x) \subseteq A_\epsilon$. Therefore the Markovian kernel restricted to set A_ϵ is $\mathsf{K}(x, A_\epsilon) = \mathsf{K}_\mathrm{m}(x, A_\epsilon) \cdot 1_{A_\epsilon^\mathrm{c}}(x) + 1_{A_\epsilon}(x)$ satisfying the preconditions of theorem B2.2.8 if $\mathsf{K}_\mathrm{m}(x, A_\epsilon) \geq \delta > 0$ for all $x \in A_\epsilon^\mathrm{c}$.

As mentioned previously, the kernel K_{m} may be interpreted as the transition probability function describing all evolutionary operators that modify the population $x \in E$ and yield the new preliminary population before the elitist operator is applied. Consider a bounded search space (notice that finite search spaces are always bounded). Assume that the mutation operator ensures that every point in the search space can be reached in one step with some minimum probability $\beta_{\mathrm{m}} > 0$ regardless of the current location. For example, the usual mutation operator for binary search spaces $\mathbb{B}^{\ell}$ has the bound $\beta_{\mathrm{m}} = \min\{p^{\ell}, (1-p)^{\ell}\} > 0$ where $p \in (0, 1)$ denotes the mutation probability. Let $\mathsf{K}_{\mathrm{cross}}$ and $\mathsf{K}_{\mathrm{mut}}$ be the Markovian kernels for crossover and mutation. Evidently, one obtains the bound $\mathsf{K}_{\mathrm{mut}}(x, \{x^*\}) \geq 1 - (1 - \beta_{\mathrm{m}})^{\mu} = \delta_{\mathrm{m}} > 0$ for the mutation kernel. It follows that the joint kernel for crossover and mutation satisfies

$$\mathsf{K}_{\mathrm{m}}(x, \{x^*\}) = \int_E \mathsf{K}_{\mathrm{cross}}(x, \mathrm{d}y)\, \mathsf{K}_{\mathrm{mut}}(y, \{x^*\}) \geq \delta_{\mathrm{m}} \int_E \mathsf{K}_{\mathrm{cross}}(x, \mathrm{d}y) = \delta_{\mathrm{m}} \mathsf{K}_{\mathrm{cross}}(x, E) = \delta_{\mathrm{m}} > 0$$

which in turn implies that this type of mutation and elitist selection leads to global convergence regardless of the chosen crossover operator.

If the search space is not bounded the argumentation is different, but it is still possible to derive positive bounds for the joint Markovian kernel for many combinations of crossover and mutation operators. See the article by Rudolph (1996) for further examples.

Finally, consider the theory of supermartingales in order to obtain global convergence results. Principally one has to calculate

$$\mathsf{E}[f(b(X_{t+1})) \mid X_t] = \int_E f(b(y)) \cdot \mathsf{P}\{X_{t+1} \in \mathrm{d}y \mid X_t\}. \tag{B2.2.5}$$

If $\mathsf{E}[f(b(X_{t+1})) \mid X_t] \leq f(b(X_t))$ almost surely for all $t \geq 0$ and the conditional expectation in (B2.2.5) exists, then the sequence $(f(b(X_t)) - f^* : t \geq 0)$ is a nonnegative supermartingale. In fact, it suffices to calculate

$$\mathsf{E}[f(b(X_{t+1})) \mid X_t = x] = \int_E f(b(y)) \cdot \mathsf{K}(x, \mathrm{d}y)$$

and to compare this expression with $f(b(x))$. Then theorem B2.2.7 may be useful to prove global convergence and to obtain bounds on the convergence rates. This topic is treated in more detail in Section B2.4. B2.4

References

Davis T E 1991 *Toward an Extrapolation of the Simulated Annealing Convergence Theory onto the Simple Genetic Algorithm* PhD Thesis, University of Florida at Gainesville

Davis T E and Principe J 1993 A Markov chain framework for the simple genetic algorithm *Evolut. Comput.* **1** 269–88

Doob J L 1967 *Stochastic Processes* 7th edn (New York: Wiley)

Fogel D B 1994 Asymptotic convergence properties of genetic algorithms and evolutionary programming: analysis and experiments *Cybernet. Syst.* **25** 389–407

Goodman R 1988 *Introduction to Stochastic Models* (Menlo Park, CA: Benjamin–Cummings)

Iosifescu M 1980 *Finite Markov Processes and Their Applications* (Chichester: Wiley)

Isaacson D L and Madsen R W 1976 *Markov Chain Theory and Applications* (New York: Wiley)

Minc H 1988 *Nonnegative Matrices* (New York: Wiley)

Neveu J 1975 *Discrete-Parameter Martingales* (Amsterdam: North-Holland)

Nix A E and Vose M D 1992 Modeling genetic algorithms with Markov chains *Ann. Math. Artificial Intell.* **5** 79–88

Nummelin E 1984 *General Irreducible Markov Chains and Non-negative Operators* (Cambridge: Cambridge University Press)

Rosenthal J S 1995 Convergence rates for Markov chains *SIAM Rev.* **37** 387–405

Rudolph G 1994a Convergence of non-elitist strategies *Proc. 1st IEEE Conf. on Computational Intelligence* vol 1 (Piscataway, NJ: IEEE) pp 63–6

——1994b Convergence properties of canonical genetic algorithms *IEEE Trans. Neural Networks* **NN-5** 96–101

——1996 Convergence of evolutionary algorithms in general search spaces *Proc. 3rd IEEE Conf. on Evolutionary Computation* (Piscataway, NJ: IEEE) pp 50–4

Seneta E 1981 *Non-negative Matrices and Markov Chains* 2nd edn (New York: Springer)

Suzuki J 1995 A Markov chain analysis on simple genetic algorithms *IEEE Trans. Syst. Man Cybernet.* **SMC-25** 655–9

Further reading

1. Williams D 1991 *Probability with Martingales* (Cambridge: Cambridge University Press)

   A lively introduction to the theory of (super-) martingales. It also links supermartingales with potential theory and Markov chains (pp 103–5).

2. Bucy R S 1965 Stability and positive supermartingales *J. Differential Equations* **1** 151–5

   This article establishes the connection between the convergence of nonnegative supermartingales and the concept of Liapunov stability of dynamical stochastic systems.

3. Robbins H and Siegmund D 1971 A convergence theorem for non negative almost supermartingales and some applications *Optimizing Methods in Statistics* ed J Rustagi (New York: Academic) pp 233–57

   Robbins and Siegmund provide convergence theorems for stochastic processes that satisfy a weaker version of the supermartingale condition.

4. Rudolph G and Sprave J 1995 A cellular genetic algorithm with self-adjusting acceptance threshold *Proc. 1st IEE/IEEE Int. Conf. on Genetic Algorithms in Engineering Systems: Innovations and Applications* (London: IEE) pp 365–72

   These authors model evolutionary algorithms with spatial structure in the framework of probabilistic automata networks.

5. Qi X and Palmieri F 1994 Theoretical analysis of evolutionary algorithms with infinite population size in continuous space, part I: basic properties *IEEE Trans. Neural Networks* **NN-5** 102–19

   EAs with search space $\mathbb{R}^n$ are investigated under the assumption that the population size is infinite. Under this scenario the theory becomes much simpler than for finite population size, but the qualitative behavior of these evolutionary 'algorithms' is quite different from that of true EAs. In fact, the expected absorption time would be of order O(1). This objection also holds for the next three articles.

6. Vose M D and Liepins G E 1991 Punctuated equilibria in genetic search *Complex Syst.* **5** 31–44

7. Vose M D and Wright A H 1995 Simple genetic algorithms with linear fitness *Evolut. Comput.* **2** 347–68

8. Whitley L D 1993 An executable model of a simple genetic algorithm *Foundations of Genetic Algorithms 2* ed L D Whitley (San Mateo, CA: Morgan Kaufmann) pp 45–62

B2.3 Modes of stochastic convergence

Günter Rudolph

Abstract

The purpose of this section is to introduce the notion of stochastic convergence of sequences of random variables and to present some interrelationships between various modes of stochastic convergence. Building on this foundation a precise definition of global convergence of evolutionary algorithms is given.

The term *convergence* is used in classical analysis to describe the limit behavior of numerical deterministic sequences. It is natural to expect that a similar concept ought exist for random sequences. In fact, such a concept does exist—but there is a difference: since random sequences are defined on probability spaces the main difference between the convergence concept of classical analysis and stochastic convergence relies on the fact that the latter must take into account the existence of a probability measure. As a consequence, depending on the manner in which the probability measure enters the definition various modes of stochastic convergence must be distinguished.

Definition B2.3.1. Let X be a random variable and $(X_t : t \geq 0)$ a sequence of random variables defined on a probability space $(\Omega, \mathcal{A}, \mathsf{P})$. Then (X_t) is said:

(i) to *converge completely* to X, denoted as $X_t \xrightarrow{\text{c}} X$, if for any $\epsilon > 0$

$$\lim_{t\to\infty} \sum_{i=0}^{t} \mathsf{P}\{|X_i - X| > \epsilon\} < \infty \tag{B2.3.1}$$

(ii) to *converge almost surely* to X, denoted as $X_t \xrightarrow{\text{a.s.}} X$, if

$$\mathsf{P}\left\{\lim_{t\to\infty} |X_t - X| = 0\right\} = 1$$

(iii) to *converge in probability* to X, denoted as $X_t \xrightarrow{\mathsf{P}} X$, if for any $\epsilon > 0$

$$\lim_{t\to\infty} \mathsf{P}\{|X_t - X| > \epsilon\} = 0 \tag{B2.3.2}$$

(iv) to *converge in mean* to X, denoted as $X_t \xrightarrow{\text{m}} X$, if

$$\lim_{t\to\infty} \mathsf{E}[|X_t - X|] = 0.$$

□

Some interrelations between these concepts are summarized below.

Theorem B2.3.1 (Lukacs 1975, pp 33–6, 51–2, Chow and Teicher 1978, pp 43–4). Let X be a random variable and $(X_t : t \geq 0)$ a sequence of random variables defined on a probability space $(\Omega, \mathcal{A}, \mathsf{P})$. The following implications are valid:

$$X_t \xrightarrow{\text{c}} X \Rightarrow X_t \xrightarrow{\text{a.s.}} X \Rightarrow X_t \xrightarrow{\mathsf{P}} X \text{ and } X_t \xrightarrow{\text{m}} X \Rightarrow X_t \xrightarrow{\mathsf{P}} X.$$

The reverse implications are not true in general. But if Ω is countable then convergence in probability is equivalent to almost sure convergence. □

Evidently, if the probabilities in (B2.3.2) converge to zero sufficiently fast that the series in (B2.3.1) is finite, then convergence in probability implies complete convergence, but which additional conditions must be fulfilled such that some of the first three modes of convergence given in definition B2.3.1 imply convergence in mean? In other words, when may one interchange the order of taking a limit and expectation such that

$$\lim_{t\to\infty} \mathsf{E}[X_t] = \mathsf{E}[\lim_{t\to\infty} X_t]?$$

To answer the question one has to introduce the notion of *uniform integrability* of random variables.

Definition B2.3.2. A collection of random variables $(X_t : t \geq 0)$ is called *uniformly integrable* if

$$\sup\{\, \mathsf{E}[|X_t|] : t \geq 0 \,\} < \infty$$

and for every $\epsilon > 0$ there exists a $\delta > 0$ such that $\mathsf{P}\{A_t\} < \delta$ implies $|\mathsf{E}[X_t \cdot 1_{A_t}]| < \epsilon$ for every $t \geq 0$. □

Now the following result is provable.

Theorem B2.3.2 (Chow and Teicher 1978, p 100). A sequence of random variables converges in mean if and only if the sequence is uniformly integrable and converges in probability. □

Since the defining condition of uniform integrability is rather unwieldly, sufficient but simpler conditions are often useful.

Theorem B2.3.3 (Williams 1991, pp 127–8). Let Y be a nonnegative random variable and $(X_t : t \geq 0)$ be a collection of random variables on a joint probability space. If $|X_t| < Y$ for all $n \geq 0$ and $\mathsf{E}[Y] < \infty$ then the random variables $(X_t : t \geq 0)$ are uniformly integrable. □

Evidently, the above result remains valid if the random variable Y is replaced by some nonnegative finite constant. Another useful convergence condition is given below.

Theorem B2.3.4 (Chow and Teicher 1978, pp 98–9). If $(X_t : t \geq 0)$ are random variables with $\mathsf{E}[|X_t|] < \infty$ and

$$\lim_{t\to\infty} \sup_{s>t} \mathsf{E}[|X_s - X_t|] = 0$$

there exists a random variable X with $\mathsf{E}[|X|] < \infty$ such that $X_t \xrightarrow{\text{m}} X$ and conversely. □

The last mode of stochastic convergence considered here is related to convergence of distribution functions.

Definition B2.3.3. Let $\{F_X(x), F_{X_t}(x) : t \geq 0\}$ be a collection of distribution functions of random variables X and $(X_t : t \geq 0)$ on a probability space $(\Omega, \mathcal{A}, \mathsf{P})$. If

$$\lim_{t\to\infty} F_{X_t}(x) = F_X(x)$$

for every continuity point x of $F_X(\cdot)$, then the sequence F_{X_t} is said to *converge weakly* to F_X, denoted as $F_{X_t} \xrightarrow{\text{w}} F_X$. In such an event, the sequence of random variables $(X_t : t \geq 0)$ is said to *converge in distribution* to X, denoted as $X_t \xrightarrow{\text{d}} X$. □

This concept has a simple relationship to convergence in probability.

Theorem B2.3.5 (Lukacs 1975, p 33, 38). Let X and $(X_t : t \geq 0)$ be random variables on a joint probability space. Then $X_t \xrightarrow{\mathsf{P}} X \Rightarrow X_t \xrightarrow{\text{d}} X$. Conversely, if $X_t \xrightarrow{\text{d}} X$ and F_X is degenerated (i.e. X is a constant) then $X_t \xrightarrow{\mathsf{P}} X$. □

After these preparatory statements one is in the position to establish the connection between stochastic convergence of random variables and the term *global convergence* of evolutionary algorithms. For this purpose let A_x be the object variable space of the *optimization problem* B2.1

$$\min\{f(x) : x \in A_x\} \qquad \text{resp.} \qquad \max\{f(x) : x \in A_x\}$$

where $f : A_x \to \mathbb{R}$ is the objective function. An individual is an element of the space $I = A_x \times A_s$ where A_s is the (possibly empty) space of strategy parameters. Thus, the population P_t of individuals at generation $t \geq 0$ of some evolutionary algorithm is an element of the product space I^μ where μ is the size of the parent population. Since the genetic operators are stochastic the sequence $(P_t : t \geq 0)$ generated by some evolutionary algorithm (EA) is a stochastic trajectory through the space I^μ. The behavior of this trajectory, even in the limit $t \to \infty$, may be very complicated in general, but in the sense of optimization one is less interested in the behavior of this trajectory—rather, one would like to know whether or not the sequence of populations contains admissible solutions of the optimization problem that become successively better and are globally optimal in the end ideally. Therefore it suffices to observe the behavior of the trajectory of the best solution contained in populations $(P_t : t \geq 0)$. For this purpose let $b : I^\mu \to A_x$ be a map that extracts the best solution represented by some individual of a population. Thus, the stochastic sequence $(B_t : t \geq 0)$ with $B_t = b(P_t)$ is a trajectory through the space A_x. But even this stochastic sequence generally exhibits too complex a behavior to formulate a simple definition of global convergence. For example, it may oscillate between globally optimal solutions and much more complex dynamics are imaginable. To avoid these difficulties one could restrict the observations to the behavior of the sequence $(f(B_t) : t \geq 0)$ of the best objective function values contained in a population. For this purpose set $X_t = |f(b(P_t)) - f^*|$ where f^* is the global minimum or maximum of the optimization problems above. Provided that the sequence of random variables $(X_t : t \geq 0)$ converges in some mode to zero, one can be sure that the population P_t will contain better and better solutions of the optimization problem for increasing t. Therefore it appears reasonable to agree upon the following convention.

Definition B2.3.4. Let $(P_t : t \geq 0)$ be the stochastic sequence of populations generated by some evolutionary algorithm. The EA is said to *converge completely (almost surely, in probability, in mean, in distribution) to the global optimum* if the sequence $(X_t : t \geq 0)$ with $X_t = |f(b(P_t)) - f^*|$ converges completely (almost surely, in probability, in mean, in distribution) to zero. □

There are some immediate conclusions. For example, if one can show that some EA converges in distribution to the global optimum, theorem B2.3.5 ensures that the EA is globally convergent in probability. Moreover, if it is known that $|f(x)| < \infty$ for all $x \in A_x$ one may conclude, owing to theorem B2.3.3, that the EA converges in mean to the global optimum as well.

Finally, it should be remarked that the probabilistic behavior of the sequence of populations can be modeled as a *stochastic process*—in fact, in most cases these stochastic processes are *Markov chains*. B2.2, B2.2.2 Then the state space of the processes is not necessarily the product space I^μ, because the order of the individuals within a population is of no importance. However this does not affect the general concept given above—only the actual implementation of the map $b(\cdot)$ has to adjusted before *convergence properties* of evolutionary algorithms can be derived. B2.2.5

References

Chow Y S and Teicher H 1978 *Probability Theory* (New York: Springer)
Lukacs E 1975 *Stochastic Convergence* 2nd edn (New York: Academic)
Williams D 1991 *Probability with Martingales* (Cambridge: Cambridge University Press)

B2.4 Local performance measures

Hans-Georg Beyer (B2.4.1) *and Günter Rudolph* (B2.4.2)

Abstract

See the individual abstracts for sections B2.4.1 and B2.4.2.

B2.4.1 Evolution strategies and evolutionary programming

Hans-Georg Beyer

Abstract

This section provides a summary of theoretical results on the performance analysis of evolutionary algorithms (EAs), especially applicable to the evolution strategy (ES) and evolutionary programming (EP). However, the methods and paradigms presented are useful—at least in principle—for all EAs, including genetic algorithms (GAs). Performance is defined in terms of the local change of the population toward the optimum. There are different possibilities of introducing performance measures that quantify certain aspects of the approach to the optimum. Two classes of performance measures will be considered: the quality gain and the progress rate. Results on various EAs and fitness landscapes are presented and discussed in the light of basic evolutionary principles. Furthermore, the results of the progress rate analysis on the sphere model will be used to investigate the evolutionary dynamics, that is, the convergence behavior of the EA in the time domain.

B2.4.1.1 Introduction and motivation

It is important to evaluate the performance of an EA not only by empirical methods but also by theoretical analysis. Furthermore, there is a need for theoretically provable statements on *why* and *how* a specific EA works. For example, the working principle of the *recombination–crossover* operators is not yet fully understood. The benefits of recombination are very often explained by some kind of *building block hypothesis* (BBH) (Goldberg 1989). A thorough theoretical analysis of the performance behavior in *evolution strategies* (ESs) shows (Beyer 1995d), however, that a totally different explanation for the benefits of recombination holds for recombinative ESs: the so-called *genetic repair* (GR). That is, some kind of *statistical error correction* diminishing the influence of the harmful parts of the mutations in nonlinear, convex curved, fitness landscapes (for a definition of *convex curved* in GAs working on bitstrings, see Beyer (1995a, 1996b)).

C3.3 B2.5.3

The question of how and why an EA or special operators work is thus synonymous with the formulation/extraction of *basic EA principles*. Such basic principles can be extracted as the qualitative features from a theory which describes the *microscopic* behavior of the EA, i.e. the expected state change of the population from generation t to generation $t+1$, whereas the quantitative aspects of the state change are described by *(local) performance measures*.

Besides the GR principle, the *evolutionary progress principle* (EPP) and the *mutation-induced speciation by recombination* (MISR) principle have been identified in ESs. It is hypothesized (Beyer

1995a) that these three principles are also valid for other EAs, including GAs, thus building an alternative EA working paradigm which is opposed to the BBH.

Apart from these more or less philosophical questions, the usefulness of local performance measures is in three different areas. First, the influence of strategy parameters on the performance can be (analytically) investigated. This is of vital importance if the EA is to tune itself for maximal performance. Second, different genetic operators can be compared as to their (local) performance. Third, the runtime complexity of the EA can be estimated by exploiting the *macroscopic evolution* (evolutionary dynamics, convergence order) which is governed by the microscopic forces (described by the local performance measures).

This contribution is organized as follows. In the next section (B2.4.1.2) the local performance measures are defined. There are mainly two kinds of measure: the *progress rate* φ and the *quality gain* $\tilde{Q}$. From a mathematical point of view φ and $\tilde{Q}$ are functionals of the fitness function F. Thus, determining φ and $\tilde{Q}$ as functions of the strategy parameter requires the fixing of F. That is, models of the fitness landscape are to be chosen such that they can represent a wide range of fitness functions and are sufficiently simple to yield (approximate) analytical formulae for φ and $\tilde{Q}$. Such models will be presented in section B2.4.1.3. Section B2.4.1.4 is devoted to the results of the quality gain theory, whereas section B2.4.1.5 summarizes the results of the progress rate theory including multirecombinant ESs as well as ESs on noisy fitness data. Section B2.4.1.6 is devoted to dynamical aspects, i.e. the macroscopic evolution of the EAs.

B2.4.1.2 How to measure evolutionary algorithm performance

From a mathematical point of view the EA is a kind of inhomogeneous Markov process mapping the population state $\mathbf{P}(t)$ at generation t onto a new state $\mathbf{P}(t+1)$ at generation $t+1$. Generally, such processes can be described by Chapman–Kolmogorov equations (see Section B2.2). However, a direct B2.2 treatment of these integral equations is almost always excluded. Very often it is not even possible to derive analytically the transition kernel of the stochastic process. On the other hand, the full information of the Markov process is seldom really needed. Furthermore, the state density $p(\mathbf{P}(t))$ is difficult to interpret. In most cases, expectations derived from $p(\mathbf{P}(t))$ suffice. Local performance measures are defined in order to measure the expected change of certain functions of the population state $\mathbf{P}$ from generation t to $t+1$. The adjective *local* refers to the Markovian character (first-order Markov process), that is, the state at $t+1$ is fully determined by the state t. There is no $t-k$ memory with $k>0$. Thus, the evolution dynamics can be modeled by first-order difference equations (derived from the local performance measures) which can be often approximated by differential equations (see section B2.4.1.6).

Choosing the right progress measure is important, and depends on the questions to be asked about the EA under investigation. For example, if one is interested in *schema processing*, then the question B2.5 about the schema occupation numbers is the appropriate one. However, if optimization is the point of interest, or more generally *meliorization* of fitness, then the progress rate φ and the quality gain $\tilde{Q}$ are the appropriate performance measures.

The quality gain $\tilde{Q}$. As indicated by the notion of quality gain, $\tilde{Q}$ measures the expected fitness change from generation t to generation $t+1$ for the population $\mathbf{P}$ of a certain member(s) $\boldsymbol{a}_i$ of the population. If the population is considered, the $\tilde{Q}$ obtained by averaging over the whole population is also known as the *response to selection* used in quantitative genetics and introduced in GA theory by Mühlenbein and Schlierkamp-Voosen (1993). This measure is quite well suited to evaluate *proportional selection* (see C2.2 Section C2.2) and can be used for $(\mu \dot{+} \lambda)$ selection (see Section B1.3) as well. However, up to now, the B1.3 main application of $\tilde{Q}$ has been in the field of EAs with $(1 \dot{+} \lambda)$ truncation selection (as in ES and EP).

In $(1 \dot{+} \lambda)$ algorithms the λ offspring $\boldsymbol{a}_l$ are generated by mutations $\boldsymbol{z}$ from the best parent's state $\boldsymbol{y}(t)$ according to

$$\boldsymbol{a}_l(t+1) := \boldsymbol{y}(t) \oplus \boldsymbol{z}.$$

(In the case of bitstrings the XOR serves as the addition operator $\oplus$; for real or integer parameters $+$ is used instead.) Due to the one-parent procreation there is no recombination in this EA. Let us introduce the *local quality function* $Q_{\boldsymbol{y}}(\boldsymbol{x})$

$$Q_{\boldsymbol{y}(t)}(\boldsymbol{x}) := F(\boldsymbol{y}(t) \oplus \boldsymbol{x}) - F(\boldsymbol{y}(t)) \tag{B2.4.1}$$

which describes the local fitness change from the parent's fitness $F(\boldsymbol{y}(t))$ to the offspring's fitness $F(\boldsymbol{a}_l(t+1))$. The λ $\boldsymbol{a}_l(t+1)$ values can be ordered according the their local quality Q

$$Q_{1;\lambda} := Q_{\boldsymbol{y}}(\boldsymbol{a}_{1;\lambda}), Q_{2;\lambda} := Q_{\boldsymbol{y}}(\boldsymbol{a}_{2;\lambda}), \ldots, Q_{m;\lambda} := Q_{\boldsymbol{y}}(\boldsymbol{a}_{m;\lambda}), \ldots, Q_{\lambda;\lambda} := Q_{\boldsymbol{y}}(\boldsymbol{a}_{\lambda;\lambda}).$$

Here we have introduced the $m;\lambda$ nomenclature indicating the mth best offspring from the set $\{\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_\lambda\}$ with respect to their local quality Q. Note that this is a generalization of the $m:\lambda$ nomenclature often used in *order statistics* (David 1970) which indicates the nondescending ordering of λ random variates X_i $(i = 1 \ldots \lambda)$

$$X_{1:\lambda} \leq X_{2:\lambda} \leq X_{3:\lambda} \leq \ldots \leq X_{m:\lambda} \leq \ldots \leq X_{\lambda:\lambda}.$$

The *quality gain* $\bar{Q}_{1\dot{+}\lambda}$ is defined as

$$\bar{Q}_{1\dot{+}\lambda}(t) := \int Q_{\boldsymbol{y}(t)}(\boldsymbol{x}) p_{1\dot{+}\lambda}(\boldsymbol{x}) \, \mathrm{d}^n x. \tag{B2.4.2}$$

That is, $\bar{Q}_{1\dot{+}\lambda}$ is the expectation of the local quality change with respect to the $(1 \dot{+} \lambda)$ truncation selection.

Determining $\bar{Q}$ by definition (B2.4.2) is difficult. The success of the quality gain theory arises from a second approach transforming the n-dimensional integral over the parameter space domain into a one-dimensional one in the *Q picture*:

$$\bar{Q}_{1\dot{+}\lambda}(t) := \int Q p_{1\dot{+}\lambda}(Q \mid \boldsymbol{y}(t)) \, \mathrm{d}Q.$$

To be more specific, it will be assumed that the EA to be analyzed has the objective to *increase* the local quality (e.g. fitness maximizing EAs). Within the $(1, \lambda)$ selection the best offspring, i.e. the one with the highest quality Q, is chosen. Using order statistics notation one obtains

$$\bar{Q}_{1,\lambda}(t) = \mathrm{E}\left\{Q_{1;\lambda}\right\} = \mathrm{E}\left\{Q_{\lambda:\lambda}\right\} = \int Q_{\lambda:\lambda} p(Q_{\lambda:\lambda}) \, \mathrm{d}Q_{\lambda:\lambda} \tag{B2.4.3}$$

$$\bar{Q}_{1,\lambda}(t) = \int Q p_{\lambda:\lambda}(Q) \, \mathrm{d}Q. \tag{B2.4.4}$$

Here, $p_{\lambda:\lambda}(Q)$ denotes the PDF (probability density function) of the largest Q value.

In contrast to the $(1, \lambda)$ selection, in $(1+\lambda)$ algorithms the parent's quality $Q = 0$ survives if $Q_{\lambda:\lambda} < 0$ holds. Therefore, one finds for $\bar{Q}_{1+\lambda}$

$$\bar{Q}_{1+\lambda}(t) = \mathrm{E}\left\{\max\left[0, \; Q_{\lambda:\lambda}\right]\right\} = \int_{Q=0} Q p_{\lambda:\lambda}(Q) \, \mathrm{d}Q \tag{B2.4.5}$$

i.e. the only difference from equation (B2.4.4) is in the lower integration limit. The determination of $p_{\lambda:\lambda}(Q)$ is a standard task of order statistics (see e.g. David 1970). Provided that the single-mutation PDF $p(Q) := p_{1:1}(Q)$ is known, then $p_{\lambda:\lambda}(Q)$ reads

$$p_{\lambda:\lambda}(Q) = \lambda p(Q) \left[P(Q)\right]^{\lambda-1} \tag{B2.4.6}$$

with the CDF (cumulative distribution function)

$$P(Q) = \int_{Q'=-\infty}^{Q'=Q} p(Q') \, \mathrm{d}Q'. \tag{B2.4.7}$$

The central problem with this approach is to find appropriate approximations for the single-mutation density $p(Q)$. This will be discussed in section B2.4.1.4.

The progress rate φ. Unlike the quality gain $\bar{Q}$ which measures the fitness change in the one-dimensional Q space, the progress rate φ is a measure that is defined in the object parameter space. It measures the expected distance change derived from the individuals of the population and a fixed point $\hat{\boldsymbol{y}}$ (the reference point). Depending on the EA considered and on the objective to be focused on there are different possible definitions for φ. As far as function optimization is considered, choosing $\hat{\boldsymbol{y}}$ equal to that point which (globally) optimizes $F(\boldsymbol{y})$ seems to be a natural choice. Note that this choice is well defined only if the optimum state is not degenerate (multiple global optima, e.g. degenerate ground states in spin-glass models and the traveling salesman problem (TSP)). However, this problem has not been of importance so far, because the model fitness landscapes which can be treated analytically (approximations!) are very simple, having at best one optimum. Therefore, the progress rate definition

$$\varphi(t) := \mathrm{E}\left\{h\big(\hat{\boldsymbol{y}},\ \mathbf{P}(t)\big) - h\big(\hat{\boldsymbol{y}},\ \mathbf{P}(t+1)\big)\right\} \tag{B2.4.8}$$

includes all cases investigated so far. Here, $h(\cdot,\cdot)$ denotes a distance measure between the reference point $\hat{\boldsymbol{y}}$ and the population $\mathbf{P}$.

In $(1\,\dot{+}\,\lambda)$ algorithms the h measure becomes

$$h\big(\hat{\boldsymbol{y}},\ \mathbf{P}(t)\big) = h\big(\hat{\boldsymbol{y}},\ \boldsymbol{y}(t)\big) \qquad h\big(\hat{\boldsymbol{y}},\ \mathbf{P}(t+1)\big) = h\big(\hat{\boldsymbol{y}},\ \boldsymbol{a}_{1;\lambda}(t+1)\big) \tag{B2.4.9}$$

and as far as real-valued parameter spaces are concerned the Euclidean norm can serve as distance measure. Thus, φ reads

$$\varphi_{1\dot{+}\lambda}(t) := \mathrm{E}\left\{\|\hat{\boldsymbol{y}} - \boldsymbol{y}(t)\| - \|\hat{\boldsymbol{y}} - \boldsymbol{a}_{1;\lambda}(t+1)\|\right\}. \tag{B2.4.10}$$

If multiparent algorithms are considered, especially of $(\mu\ \dot{+}\ \lambda)$ selection type, average distance measures will be used. Let the parent states be $\boldsymbol{a}_{m;\lambda}(t)$ at generation t and $\boldsymbol{a}_{m;\lambda}(t+1)$ at $t+1$, then $\varphi_{\mu\dot{+}\lambda}$ can be defined as

$$\varphi_{\mu\dot{+}\lambda}(t) := \mathrm{E}\left\{\frac{1}{\mu}\sum_{m=1}^{\mu}\|\hat{\boldsymbol{y}} - \boldsymbol{a}_{m;\lambda}(t)\| - \frac{1}{\mu}\sum_{m=1}^{\mu}\|\hat{\boldsymbol{y}} - \boldsymbol{a}_{m;\lambda}(t+1)\|\right\} \tag{B2.4.11}$$

$$\varphi_{\mu\dot{+}\lambda}(t) = \frac{1}{\mu}\sum_{m=1}^{\mu}\mathrm{E}\left\{\|\hat{\boldsymbol{y}} - \boldsymbol{a}_{m;\lambda}(t)\| - \|\hat{\boldsymbol{y}} - \boldsymbol{a}_{m;\lambda}(t+1)\|\right\}. \tag{B2.4.12}$$

This definition will be used to evaluate the performance of the (μ,λ) ES on the spherical model (see section B2.4.1.5).

Apart from definition (B2.4.12) there is another possibility to introduce a collective distance measure with respect to the *center of mass* individual

$$\langle\boldsymbol{a}\rangle_{\mu}(t) := \frac{1}{\mu}\sum_{m=1}^{\mu}\boldsymbol{a}_{m;\lambda}(t). \tag{B2.4.13}$$

Especially, if recombinative EAs are under investigation this will be the appropriate measure leading to the φ definition for (multi)recombinant $(\mu/\rho\,\dot{+}\,\lambda)$ ESs

$$\varphi_{\mu/\rho\dot{+}\lambda}(t) := \mathrm{E}\left\{\|\hat{\boldsymbol{y}} - \langle\boldsymbol{a}\rangle_{\mu}(t)\| - \|\hat{\boldsymbol{y}} - \langle\boldsymbol{a}\rangle_{\mu}(t+1)\|\right\}. \tag{B2.4.14}$$

It is quite clear that definition (B2.4.11) and (B2.4.14) are not equivalent. However, for *species-like* populations crowded around a *wild-type parent* —the center of mass individual—the two progress rates become comparable, if the distance of the population from the reference point $\hat{\boldsymbol{y}}$ is large compared to the spreading of the population.

The normal progress φ_{R}. The derivation of (analytical) results for the progress rate φ is generally very difficult even for $(1\,\dot{+}\,\lambda)$ ESs, whereas the treatment of the $\bar{Q}$ measure on $(1\,\dot{+}\,\lambda)$ strategies is easier to accomplish (even for correlated mutations). This has led to a progress rate definition (Rechenberg 1994) which measures the distance from the hypersurface $Q_{\boldsymbol{y}(t)}(\boldsymbol{x}) = 0$ at $\boldsymbol{x} = \mathbf{o}$ to the hypersurface $Q_{\boldsymbol{y}(t)}(\boldsymbol{x}) = \bar{Q}_{1\dot{+}\lambda}$ in gradient direction $\nabla Q_{\boldsymbol{y}(t)}(\boldsymbol{x})\big|_{\boldsymbol{x}=\boldsymbol{o}}$. This progress rate will be called *normal progress*

φ_R, because the gradient direction of $Q_{\boldsymbol{y}(t)}(\boldsymbol{x})$ is normal to the hypersurface $Q_{\boldsymbol{y}(t)}(\boldsymbol{x}) = 0$. By Taylor expansion one finds

$$\bar{Q}_{1^+_\lambda}(t) \approx \nabla Q_{\boldsymbol{y}(t)}(\boldsymbol{x})\big|_{\boldsymbol{x}=\boldsymbol{o}}\ \delta\boldsymbol{x} \ =:\ \nabla Q_{\boldsymbol{y}(t)}(\boldsymbol{x}) \left(\frac{\nabla Q_{\boldsymbol{y}(t)}(\boldsymbol{x})}{\left\| \nabla Q_{\boldsymbol{y}(t)}(\boldsymbol{x}) \right\|}\ \varphi_R \right)$$

and therefore, the normal progress definition becomes

$$\varphi_R(t) := \frac{\bar{Q}_{1^+_\lambda}(t)}{\left\| \nabla Q_{\boldsymbol{y}(t)}(\boldsymbol{x})\big|_{\boldsymbol{x}=\boldsymbol{o}} \right\|}. \tag{B2.4.15}$$

The normal progress φ_R can be used to obtain an estimate for φ from the 'hard' definition (B2.4.10). However, due to the simple definition (B2.4.15) one should not expect a 'higher information content' than given by the quality gain $\bar{Q}$.

Definition (B2.4.15) is just a local normalization for $\bar{Q}$. Only for fitness landscapes $Q_{\boldsymbol{y}(t)}(\boldsymbol{x}) = 0$ which are nearly symmetrical at $\boldsymbol{x} = \mathbf{o}$ and where $\hat{\boldsymbol{y}}$ is located in the vicinity of the symmetry axis (equal to the direction of the local gradient $\nabla Q_{\boldsymbol{y}(t)}(\boldsymbol{x})\big|_{\boldsymbol{x}=\boldsymbol{o}}$) does the φ_R concept deliver results in accordance with the 'hard' progress definition (B2.4.10) (see Beyer 1994a).

B2.4.1.3 Models of fitness landscapes

The determination of φ and $\bar{Q}$ depends on the *fitness landscape* and on the search operators acting on these landscapes. In order to derive analytical approximations for φ and $\bar{Q}$ one has to choose sufficiently simple fitness models. With the exception of the `OneMax` function all models to be introduced are defined in a real-valued parameter space of dimension n.

The sphere model. The most prominent model is the *hypersphere*. The 'equifitness' values $F(\boldsymbol{y}) = c$, where c is a constant, build concentric hyperspheres around the optimum point $\hat{\boldsymbol{y}}$ with fitness value $\hat{F}$

$$F(\boldsymbol{y}) = \hat{F} - F\left(\left\|\boldsymbol{y} - \hat{\boldsymbol{y}}\right\|\right) = \hat{F} - F(\|\boldsymbol{r}\|) = \hat{F} - F(r)$$

and the radius vector $\boldsymbol{r} := \boldsymbol{y} - \hat{\boldsymbol{y}}$ of length $r := \|\boldsymbol{r}\|$. It is assumed that $F(r)$ is a monotonic function of r and $F(r = 0) = 0$. The local quality $Q_{\boldsymbol{y}}(\boldsymbol{x})$ thus becomes (cf equation (B2.4.1))

$$Q_{\boldsymbol{y}}(\boldsymbol{x}) = F(\|\boldsymbol{r}\|) - F(\|\boldsymbol{r} + \boldsymbol{x}\|) = F(r) - F\left(\|\tilde{\boldsymbol{r}}\|\right) = F(r) - F(\tilde{r}) =: Q_r(\tilde{r})$$

with the offspring's radius vector $\tilde{\boldsymbol{r}} := \boldsymbol{r} + \boldsymbol{x}$ of length $\tilde{r} := \|\tilde{\boldsymbol{r}}\|$.

The inclined (hyper)plane. From the sphere model one can easily derive the linear model *inclined (hyper)plane*. Under the condition that the local radius r is much larger than the generational change $\|\boldsymbol{x}\|$, i.e. provided that $\|\boldsymbol{r}\| \gg \|\boldsymbol{x}\|$ holds, then $Q_{\boldsymbol{y}}(\boldsymbol{x})$ can be expanded into a Taylor series breaking off after the linear term

$$Q(\boldsymbol{x}) = F(\|\boldsymbol{r}\|) - F(\|\boldsymbol{r} + \boldsymbol{x}\|) \approx F(\|\boldsymbol{r}\|) - \left(F(\|\boldsymbol{r}\|) + \frac{\mathrm{d}F}{\mathrm{d}r}\ \nabla\|\boldsymbol{r}\|\boldsymbol{x} \right)$$

$$Q(\boldsymbol{x}) \approx -\frac{\mathrm{d}F}{\mathrm{d}r} \frac{\boldsymbol{r}^{\mathrm{T}}}{\|\boldsymbol{r}\|}\boldsymbol{x} \ =\ \frac{\mathrm{d}F}{\mathrm{d}r} \left(-\boldsymbol{e}_r^{\mathrm{T}}\right) \boldsymbol{x} \ =:\ \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x}.$$

Here $\boldsymbol{e}_r := \boldsymbol{r}/\|\boldsymbol{r}\|$ has been introduced as the unity vector in the $\boldsymbol{r}$ direction.

The advantage of this approach arises from the possibility of deriving progress rates φ for the inclined plane from those of the sphere by applying the $\|\boldsymbol{r}\| \gg \|\boldsymbol{z}\|$ condition in the φ formula of the sphere model. That is, the φ formulae can be obtained from the sphere by assuming small standard deviations σ of the $\boldsymbol{z}$ mutations (see section B2.4.1.5).

Models with constraints: corridor and discus. The *corridor model* introduced by Rechenberg (see Rechenberg 1994) defines a success domain on an inclined hyperplane which is shaped like a narrow corridor $\mathcal{C}$

$$\texttt{corridor} \qquad \mathcal{C}: \quad \begin{cases} -\infty \le x_1 \le \infty \\ -b \le x_i \le b \qquad 1 < i \le n. \end{cases} \tag{B2.4.16}$$

The quality function $Q(\boldsymbol{x})$ reads

$$Q(\boldsymbol{x}) := \begin{cases} q_{\mathcal{C}}(x_1) & \boldsymbol{x} \in \mathcal{C} \\ \texttt{lethal} & \boldsymbol{x} \notin \mathcal{C} \end{cases} \tag{B2.4.17}$$

where `lethal` may be $+\infty$ (minimization) or $-\infty$ (maximization), respectively. Therefore, progress is only in the x_1 direction; $q_{\mathcal{C}}(x_1)$ is a monotonic function of x_1.

Unlike the corridor, in the *discus model* $\mathcal{D}$ there is only a constraint in the x_1 direction (the optimum direction) whereas the x_i $(i = 2 \ldots n)$ directions are selectively neutral,

$$\texttt{discus} \qquad \mathcal{D}: \quad \begin{cases} 0 \le x_1 \le 2b \\ -a \le x_i \le a \qquad 1 < i \le n \end{cases} \tag{B2.4.18}$$

with

$$\texttt{discus condition:} \qquad b \ll a. \tag{B2.4.19}$$

The quality function $Q(\boldsymbol{x})$ reads

$$Q(\boldsymbol{x}) := \begin{cases} q_{\mathcal{D}}(x_1) & \boldsymbol{x} \in \mathcal{D} \\ \texttt{lethal} & \boldsymbol{x} \notin \mathcal{D} \end{cases} \tag{B2.4.20}$$

with $q_{\mathcal{D}}(x_1)$ having its optimum at $x_1 = \hat{x}_1 = b$ and the boundary condition $q_{\mathcal{D}}(0) = q_{\mathcal{D}}(2b) = 0$.

General quadratic and higher-order fitness landscapes. Especially for the $\bar{Q}$ measure but also for the *mean-radius* differential geometry approach (see section B2.4.1.5) fitness models can be used which are obtained by local Taylor expansion of equation (B2.4.1). This leads to the general quadratic model

$$Q_{\boldsymbol{y}(t)}(\boldsymbol{x}) := \boldsymbol{b}^{\mathrm{T}}(t)\,\boldsymbol{x} - \boldsymbol{x}^{\mathrm{T}}\mathbf{Q}(t)\,\boldsymbol{x} \tag{B2.4.21}$$

with

$$(\boldsymbol{b}(t))_i := \left.\frac{\partial F(\boldsymbol{y})}{\partial y_i}\right|_{\boldsymbol{y}=\boldsymbol{y}(t)} \qquad (\mathbf{Q}(t))_{ij} := -\frac{1}{2}\left.\frac{\partial^2 F(\boldsymbol{y})}{\partial y_i\,\partial y_j}\right|_{\boldsymbol{y}=\boldsymbol{y}(t)} \qquad (i, j = 1 \ldots n). \tag{B2.4.22}$$

In cases of vanishing Hessian matrix $\mathbf{Q}$ it even can be necessary to use higher-order derivatives. As an example the fitness model

$$Q_{\boldsymbol{y}(t)}(\boldsymbol{x}) := \sum_{i=1}^{n} b_i(t)x_i - \sum_{i=1}^{n} c_i(t)(x_i)^4 \tag{B2.4.23}$$

will be considered.

Further fitness models are imaginable; however, the current analysis has been performed for equations (B2.4.21) and (B2.4.23) only.

The bit counting function `OneMax`. The `OneMax` function simply counts the number of bits. Let $\boldsymbol{a} = (a_1, \ldots a_\ell)$ be a bitstring of length ℓ with $a_i \in \{0, 1\}$, then the fitness function reads

$$F(\boldsymbol{a}) := \sum_{i=1}^{\ell} a_i. \tag{B2.4.24}$$

Its maximum $\hat{F} = \ell$ is obtained for $a_i \equiv 1$ $(i = 1 \ldots \ell)$. The local quality function can be expressed as

$$Q_{\boldsymbol{a}(t)}(\boldsymbol{a}(t+1)) = F(\boldsymbol{a}(t+1)) - F(\boldsymbol{a}(t)) =: F(t+1) - F(t). \tag{B2.4.25}$$

`OneMax`, equation (B2.4.24), plays a similar role as $F(\boldsymbol{y}) = -[(\boldsymbol{y})^2]^{1/2}$ (a special sphere model) for the real-valued EAs, because $\varphi_{1^+\lambda} = \bar{Q}_{1^+\lambda}$ holds. In this case $\varphi_{1^+\lambda}$ can be interpreted as the average change of the Hamming distance toward the optimum.

Noisy fitness landscapes. Optimization tasks in engineering science as well as computer simulations can be affected by noise. That is, the measured fitness value is disturbed by random fluctuations. Noise can mislead the evolutionary search considerably; in particular the selection process is deceived (see below). Therefore, it is important to include simple noise models in the performance analysis. This has been done for $(\tilde{1} \dagger \tilde{\lambda})$ strategies (the tildes above the 1 and the λ indicate that the parent's and the offspring's fitness determination is disturbed by noise). The noise model is a local one which assumes Gaussian fluctuations. The measured local quality $\tilde{Q}$ is modeled as

$$\tilde{Q}_{\boldsymbol{y}^{(t)}}(\boldsymbol{x}) := Q_{\boldsymbol{y}^{(t)}}(\boldsymbol{x}) + \epsilon_Q(\boldsymbol{y}(t)) \tag{B2.4.26}$$

where the fluctuation term $\epsilon_Q(\boldsymbol{y}(t))$ is normally distributed

$$p(\epsilon_Q) = \frac{1}{(2\pi)^{1/2}\,\sigma_{\epsilon_Q}} \exp\left[-\frac{1}{2}\left(\frac{\epsilon_Q}{\sigma_{\epsilon_Q}}\right)^2\right] \qquad \text{with} \qquad \sigma_{\epsilon_Q} = \sigma_{\epsilon_Q}(\boldsymbol{y}(t)). \tag{B2.4.27}$$

The noise strength σ_{ϵ_Q} depends in general on the local parental state $\boldsymbol{y}(t)$. Possible dependences with respect to $\boldsymbol{x}$ are neglected.

B2.4.1.4 The quality gain theory for $(1 \dagger \lambda)$ algorithms

General aspects—the single-mutation distribution. The success of the quality gain theory arises from the possibility of deriving approximations for the single-mutation CDF $P(Q)$, equation (B2.4.7). $P(Q)$ describes the distribution of the Q fluctuations generated by a *single* mutation $\boldsymbol{z}$ with mutation density $p(\boldsymbol{z})$ applied to the local quality function $Q_{\boldsymbol{y}^{(t)}}(\boldsymbol{x})$ (e.g. equations (B2.4.21), (B2.4.23), or (B2.4.25)). The single-mutation CDF $P(Q)$ depends on the local quality function and on the mutation density. The latter is assumed to be Gaussian

(i) variant: `isotropic:`
$$p(\boldsymbol{z}) = \frac{1}{(2\pi)^{n/2}}\frac{1}{\sigma^n}\exp\left(-\frac{1}{2}\frac{\boldsymbol{z}^{\mathrm{T}}\boldsymbol{z}}{\sigma^2}\right) \tag{B2.4.28}$$

(ii) variant: `correlated:`
$$p(\boldsymbol{z}) = \frac{1}{(2\pi)^{n/2}}\frac{1}{(\det\{\mathbf{C}\})^{1/2}}\exp\left(-\frac{1}{2}\boldsymbol{z}^{\mathrm{T}}\mathbf{C}^{-1}\boldsymbol{z}\right) \tag{B2.4.29}$$

in the case of real-valued parameter spaces, and concerning the `OneMax` function a single bit flipping mutation rate p_{m} for each bit a_i (cf section B2.4.1.3) is assumed

$$p(a_i(t+1) \mid a_i(t)) = p_{\mathrm{m}}\delta(a_i(t+1) - 1 + a_i(t)) \;+\; (1 - p_{\mathrm{m}})\delta(a_i(t+1) - a_i(t)). \tag{B2.4.30}$$

(Dirac's delta-function used: $\int_{-\infty}^{\infty}\delta(x-y)\,\mathrm{d}x = 1$, $\int_{-\infty}^{\infty} f(x)\delta(x-y)\,\mathrm{d}x = f(y)$.) It is quite clear that there is no general closed expression for $P(Q)$. The basic idea for a suitable approximation of $P(Q)$ is given by a series expansion of $P(Q)$ using Hermite polynomials $\mathrm{He}_k(x)$

$$\mathrm{He}_k(x) := (-1)^k \mathrm{e}^{x^2/2}\frac{\mathrm{d}^k}{\mathrm{d}x^k}\left(\mathrm{e}^{-x^2/2}\right)$$

$$\begin{aligned} &\mathrm{He}_0(x) = 1 && \mathrm{He}_1(x) = x \\ &\mathrm{He}_2(x) = x^2 - 1 && \mathrm{He}_3(x) = x^3 - 3x \\ &\mathrm{He}_4(x) = x^4 - 6x^2 + 3 && \mathrm{He}_5(x) = x^5 - 10x^3 + 15x. \end{aligned}$$

The approximated distribution function $P(Q)$ of a single mutation reads

$$P(Q) = \frac{1}{2} + \Phi_0\left(\frac{Q-m}{s}\right) - \frac{\exp -\frac{1}{2}((Q-m)/s)^2}{(2\pi)^{1/2}}\left[\frac{\kappa_3}{3!}\mathrm{He}_2\left(\frac{Q-m}{s}\right) + \left(\frac{\kappa_4}{4!}\mathrm{He}_3\left(\frac{Q-m}{s}\right) + \frac{\kappa_3^2}{72}\mathrm{He}_5\left(\frac{Q-m}{s}\right)\right) + \cdots\right] \tag{B2.4.31}$$

where $\Phi_0(\cdot)$ is the Gauss integral (Bronstein and Semendjajew 1981) which is closely related to the error function

$$\Phi_0(x) := \frac{1}{(2\pi)^{1/2}} \int_{t=0}^{t=x} \mathrm{e}^{-t^2/2}\,\mathrm{d}t = \frac{1}{2}\,\mathrm{erf}\Big(\frac{x}{2^{1/2}}\Big). \tag{B2.4.32}$$

The parameters m, s, and κ_k are the mean value m of the mutation-induced Q fluctuations, the standard deviation s of Q, and the cumulants κ_k of the standardized variate $(Q-m)/s$

$$m := \bar{Q} = \int Q(\boldsymbol{z})\,p(\boldsymbol{z})\,\mathrm{d}^n z$$

$$s := (\overline{Q^2} - \bar{Q}^2)^{1/2} \qquad \overline{Q^2} = \int (Q(\boldsymbol{z}))^2\,p(\boldsymbol{z})\,\mathrm{d}^n z$$

$$\kappa_k = \kappa_k\left\{\frac{Q-m}{s}\right\}.$$

The cumulants κ_k of a random variate X are connected with the central moments μ_k of X (see e.g. Abramowitz and Stegun 1984); for $k=3$ and $k=4$ one obtains

$$\kappa_3\{X\} = \mu_3\{X\} \qquad \kappa_4\{X\} = \mu_4\{X\} - 3\,(\mu_2\{X\})^2$$

where μ_k is defined by

$$\mu_k\{X\} := \overline{(X-\bar{X})^k}.$$

Let us present some examples where the parameters m, s, and κ_k can be analytically calculated.

Assuming correlated mutations, equation (B2.4.29), with covariance matrix $\mathbf{C}$ and the quadratic model (B2.4.21) one finds up to κ_4

$$m = -\mathrm{Tr}\{\mathbf{QC}\} \qquad s = \left(\boldsymbol{b}^{\mathrm{T}}\mathbf{C}\boldsymbol{b} + 2\,\mathrm{Tr}\{(\mathbf{QC})^2\}\right)^{1/2} \tag{B2.4.33}$$

$$\kappa_3 = -\frac{6\,\boldsymbol{b}^{\mathrm{T}}\mathbf{CQC}\boldsymbol{b} + 8\,\mathrm{Tr}\{(\mathbf{QC})^3\}}{\left(\boldsymbol{b}^{\mathrm{T}}\mathbf{C}\boldsymbol{b} + 2\,\mathrm{Tr}\{(\mathbf{QC})^2\}\right)^{3/2}} \qquad \kappa_4 = 48\,\frac{\boldsymbol{b}^{\mathrm{T}}\mathbf{CQCQC}\boldsymbol{b} + \mathrm{Tr}\{(\mathbf{QC})^4\}}{\left(\boldsymbol{b}^{\mathrm{T}}\mathbf{C}\boldsymbol{b} + 2\,\mathrm{Tr}\{(\mathbf{QC})^2\}\right)^2} \tag{B2.4.34}$$

where $\mathrm{Tr}\{\mathbf{M}\}$ is the trace of the matrix $\mathbf{M}$ (i.e. the sum over the diagonal elements of $\mathbf{M}$).

The isotropic mutation case, equation (B2.4.28), is easily obtained from the equations (B2.4.33) and (B2.4.34) by the substitution $\mathbf{C} = \sigma^2\mathbf{E}$ ($\mathbf{E}$ is the identity matrix). The results can be found in the article by Beyer (1994a).

For the local quality function (B2.4.23) the first three distribution parameters derived for isotropic Gaussian mutations (B2.4.28) are as follows:

$$m = -3\sigma^4\sum_{i=1}^{n} c_i \qquad s = \sigma\left(\sum_{i=1}^{n} b_i^2 + 96\sigma^6\sum_{i=1}^{n} c_i^2\right)^{1/2} \qquad \kappa_3 = -36\frac{\sigma^6}{s^3}\left(\sum_{i=1}^{n} b_i^2 c_i + 264\sigma^6\sum_{i=1}^{n} c_i^3\right).$$

As already pointed out, the quality gain concept can be applied to the `OneMax` function. The first two parameters m and s can be easily derived from equation (B2.4.25) by use of equation (B2.4.30). One obtains

$$m(t) = -p_{\mathrm{m}}(2F(t) - \ell) \qquad s = (\ell)^{1/2}[p_{\mathrm{m}}(1-p_{\mathrm{m}})]^{1/2}. \tag{B2.4.35}$$

If the parameters of the single-mutation CDF are determined, then one can proceed with the quality gain theory. The next section presents the $(1,\lambda)$ formula whereas the following section gives results on $(1+\lambda)$ algorithms. The quality gain theory can be extended to multiparent algorithms, provided that the parental distribution is known. This work remains to be done.

The (1,λ) formula. The quality gain formula for $(1, \lambda)$ strategies can be derived from equations (B2.4.4), (B2.4.6), and (B2.4.7) by applying the approximation (B2.4.31). One obtains the approximate $\bar{Q}_{1,\lambda}$ formula

$$\bar{Q}_{1,\lambda} = m \; + \; sc_{1,\lambda}\left(1 + \frac{10\kappa_3^2 - 9\kappa_4}{72}\right) + s\left[(d_{1,\lambda}^{(2)} - 1)\frac{\kappa_3}{6} \; - \; d_{1,\lambda}^{(3)}\frac{4\kappa_3^2 - 3\kappa_4}{72} \; + \; \ldots\right].$$

Very often the simplified variants

$$\bar{Q}_{1,\lambda} = m \; + \; sc_{1,\lambda} \; + \; s\big(d_{1,\lambda}^{(2)} - 1\big)\frac{\kappa_3}{6}$$

or even

$$\bar{Q}_{1,\lambda} = m \; + \; sc_{1,\lambda} \tag{B2.4.36}$$

suffice. These formulae contain *progress coefficients* $c_{1,\lambda}$ and $d_{1,\lambda}^{(k)}$ which are defined by

$$d_{1,\lambda}^{(k)} := \frac{\lambda}{(2\pi)^{1/2}}\int_{-\infty}^{\infty} t^k \mathrm{e}^{-t^2/2}\left[\frac{1}{2} + \Phi_0(t)\right]^{\lambda-1} \mathrm{d}t \tag{B2.4.37}$$

and

$$c_{1,\lambda} := d_{1,\lambda}^{(1)}. \tag{B2.4.38}$$

The integral (B2.4.37) is tractable for small integer λ only ($\lambda < 6$). Numerical integration has been used to obtain table B2.4.1.

Table B2.4.1. Progress coefficients for $(1, \lambda)$ strategies.

λ	$c_{1,\lambda}$	$d_{1,\lambda}^{(2)}$	$d_{1,\lambda}^{(3)}$	λ	$c_{1,\lambda}$	$d_{1,\lambda}^{(2)}$	$d_{1,\lambda}^{(3)}$
1	0.0000	1.0000	0.0000	60	2.3193	5.5856	13.970
2	0.5642	1.0000	1.4105	70	2.3774	5.8512	14.914
3	0.8463	1.2757	2.1157	80	2.4268	6.0827	15.755
4	1.0294	1.5513	2.7004	90	2.4697	6.2880	16.514
5	1.1630	1.8000	3.2249	100	2.5076	6.4724	17.207
6	1.2672	2.0217	3.7053	150	2.6492	7.1883	19.991
7	1.3522	2.2203	4.1497	200	2.7460	7.7015	22.077
8	1.4236	2.3995	4.5636	300	2.8778	8.4310	25.164
9	1.4850	2.5626	4.9512	400	2.9682	8.9524	27.457
10	1.5388	2.7121	5.3158	500	3.0367	9.3587	29.291
12	1.6292	2.9780	5.9866	600	3.0917	9.6919	30.826
14	1.7034	3.2092	6.5928	700	3.1375	9.9744	32.148
16	1.7660	3.4137	7.1464	800	3.1768	10.220	33.311
18	1.8200	3.5970	7.6565	900	3.2111	10.436	34.351
20	1.8675	3.7632	8.1298	1000	3.2414	10.630	35.292
25	1.9653	4.1210	9.1843	2000	3.4353	11.914	41.729
30	2.0428	4.4187	10.097	3000	3.5444	12.669	45.687
40	2.1608	4.8969	11.629	4000	3.6199	13.207	48.578
50	2.2491	5.2740	12.892	5000	3.6776	13.625	50.868

Example. Spherical model with local quality

$$Q(\boldsymbol{x}) = b\sum_{i=1}^{n} x_i - \sum_{i=1}^{n} x_i^2 \tag{B2.4.39}$$

and isotropic mutations $\mathbf{C} = \sigma^2\mathbf{E}$. By completing the square in equation (B2.4.39) one finds the radius r of the model (i.e. the distance to the optimum point) as well as the optimum value $\hat{Q}$ and the optimum point $\hat{x}_i$

$$r = (n)^{1/2}\frac{b}{2} \qquad \hat{Q} = n\left(\frac{b}{2}\right)^2 \qquad \hat{x}_i = \frac{b}{2}.$$

The quality gain $\bar{Q}_{1,\lambda}$ using approximation (B2.4.36) with equation (B2.4.33) by taking $\mathbf{Q} = \mathbf{E}$ and $\boldsymbol{b} = (b, b, \ldots, b)$ into account reads

$$\bar{Q}_{1,\lambda} = \sigma c_{1,\lambda} b(n)^{1/2} \left(1 + 2\left(\frac{\sigma}{b}\right)^2\right)^{1/2} - \sigma^2 n. \tag{B2.4.40}$$

If the normal progress definition (B2.4.15) is applied and $b(n)^{1/2} = 2r$, then one obtains (cf section B2.4.1.2)

$$\varphi_{\mathrm{R}} = \sigma c_{1,\lambda} \left[1 + \frac{1}{2n}\left(\sigma \frac{n}{r}\right)^2\right]^{1/2} - \frac{\sigma^2}{2}\frac{n}{r}.$$

Example. `OneMax`: with equations (B2.4.36) and (B2.4.35) the quality gain becomes

$$\bar{Q}_{1,\lambda} = [p_{\mathrm{m}}(1 - p_{\mathrm{m}})]^{1/2} \ell^{1/2} c_{1,\lambda} - p_{\mathrm{m}}(2F(t) - \ell). \tag{B2.4.41}$$

Remark B2.4.1. The quality of the approximations used depends on n and ℓ, respectively. They are asymptotically exact ($n, \ell \to \infty$).

Remark B2.4.2. The formulae (B2.4.40) and (B2.4.41) support the *EPP hypothesis*. The EPP (*evolutionary progress principle*, Beyer 1995a, 1996b) states that the evolutionary progress (or the quality gain) is the result of two opposite tendencies, the *progress loss* and the *progress gain*.

The *progress loss* in approximation (B2.4.36), is due to the m part, i.e. the expected, mutation-induced Q change. The larger σ and p_{m}, the larger the progress loss will be (NB, this holds for equation (B2.4.41) if $F(t) > \ell/2$).

The *progress gain* is associated with the $sc_{1,\lambda}$ term which describes the influence of selection. Note $c_{1,1} = 0$; that is, in the case of one offspring, there is no progress gain at all. The progress gain depends on the mutation strength σ and the mutation rate p_{m}.

Because of the opposite tendencies of progress loss and gain and their dependence on σ and p_{m}, there must be a locally optimal σ and p_{m} value, respectively, that maximizes the quality gain. *Self-adaptation*, C7.1 as used in ES (Schwefel 1995) and EP (Fogel 1992), is aiming at the self-tuning which drives the algorithm into the locally optimal mutation strength σ or mutation rate p_{m}.

The (1+λ) formula. The derivation technique for the quality gain formula on $(1 + \lambda)$ algorithms (ES and EP) is similar to that of the $(1, \lambda)$ formula. The only difference is in the lower limit of integral (B2.4.5). However, this makes the derivation of analytical expressions difficult. The approximation result is

$$\begin{aligned}\bar{Q}_{1+\lambda} = {} & \left[m - s\frac{\kappa_3}{6} + \ldots\right]\left[1 - (P(0))^{\lambda}\right] \\ & + s\left(1 + \frac{10\kappa_3^2 - 9\kappa_4}{72} + \ldots\right) d_{1+\lambda}^{(1)}\left(\Phi_0^{-1}\left(P(0) - \frac{1}{2}\right)\right) \\ & + s\left(\frac{\kappa_3}{6} + \ldots\right) d_{1+\lambda}^{(2)}\left(\Phi_0^{-1}\left(P(0) - \frac{1}{2}\right)\right) \\ & - s\left(\frac{4\kappa_3^2 - 3\kappa_4}{72} + \ldots\right) d_{1+\lambda}^{(3)}\left(\Phi_0^{-1}\left(P(0) - \frac{1}{2}\right)\right) - \ldots.\end{aligned} \tag{B2.4.42}$$

Here, $P(0)$ is given by equation (B2.4.31) (let $Q = 0$), $\Phi_0^{-1}(\cdot)$ is the inverse function to the Gauss integral (B2.4.32) (note that $\Phi_0^{-1}(y) = 2^{1/2}\,\mathrm{erf}^{-1}(2y)$ holds), and $d_{1+\lambda}^{(k)}(x)$ are the so-called progress functions

$$d_{1+\lambda}^{(k)}(x) := \frac{\lambda}{(2\pi)^{1/2}} \int_{t=x}^{t=\infty} t^k \mathrm{e}^{-t^2/2} \left[\frac{1}{2} + \Phi_0(t)\right]^{\lambda - 1} \mathrm{d}t. \tag{B2.4.43}$$

The difficulties of obtaining analytical $\bar{Q}_{1+\lambda}$ approximations are because of the (likely) intractability of the integral (B2.4.43) for $\lambda > 2$. The results for $\lambda = 1$ are

$$d_{1+1}^{(1)}(x) = \frac{\mathrm{e}^{-x^2/2}}{(2\pi)^{1/2}} \qquad d_{1+1}^{(2)}(x) = \frac{1}{2} - \Phi_0(x) + x\frac{\mathrm{e}^{-x^2/2}}{(2\pi)^{1/2}} \qquad d_{1+1}^{(3)}(x) = \frac{2 + x^2}{(2\pi)^{1/2}}\mathrm{e}^{-x^2/2} \tag{B2.4.44}$$

and for $\lambda = 2$

$$d^{(1)}_{1+2}(x) = \frac{1}{(\pi)^{1/2}}\left(\frac{1}{2} - \Phi_0(2^{1/2}x)\right) + \frac{2}{(2\pi)^{1/2}}\left(\frac{1}{2} + \Phi_0(x)\right)\mathrm{e}^{-x^2/2} \tag{B2.4.45}$$

$$d^{(2)}_{1+2}(x) = 1 - \left(\frac{1}{2} + \Phi_0(x)\right)^2 + \frac{2x}{(2\pi)^{1/2}}\left(\frac{1}{2} + \Phi_0(x)\right)\mathrm{e}^{-x^2/2} + \frac{1}{2\pi}\mathrm{e}^{-x^2}$$

$$d^{(3)}_{1+2}(x) = \frac{1}{\pi^{1/2}}\frac{5}{2}\left(\frac{1}{2} - \Phi_0(2^{1/2}x)\right) + \frac{2}{(2\pi)^{1/2}}(2+x^2)\left(\frac{1}{2} + \Phi_0(x)\right)\mathrm{e}^{-x^2/2} + \frac{x}{2\pi}\mathrm{e}^{-x^2}.$$

B2.4.1.5 The progress rate theory

Most effort in ES theory has been focused on the calculation of progress rates. There is a wealth of results from different decades and of different approximation quality.

In order to have a certain system, results on the nonspherical corridor and discus models will be presented first. In the subsequent section some preparations for the sphere model will be given: the 'n/r normalization' and the differential geometry approach which allows the introduction of a mean radius of curvature on nonspherical (sufficiently smooth) fitness landscapes. The next section is devoted to $(\mu \dot{+} \lambda)$ algorithms in the asymptotic $n \to \infty$ limit, whereas the following two sections present results for $(\mu/\mu, \lambda)$ multirecombinant evolution strategies and the last of these sections B2.4.1.5 provides φ formulae on noisy fitness data.

Note B2.4.1. All progress rate formulae derived so far assume isotropic Gaussian mutations (B2.4.28). Results for the general case, equation (B2.4.29), can be obtained by the normal progress approach (cf section B2.4.1.2); however, one should keep in mind that this approach is not equivalent to the 'hard' progress rate definition (section B2.4.1.2) and therefore can produce unsatisfactory and inexact results.

Note B2.4.2. The results concerning $(+)$ selection (elitist strategies, see Section C2.7.4) derived for the ES hold *mutatis mutandis* for EP (Fogel 1994, 1995). C2.7.4

The corridor and the discus. Corridor and discus are 'early' models of the progress rate theory in the sense that they have not received further investigations since 1990 (or earlier).

The corridor model was treated for the $(1+1)$ algorithm by Rechenberg in the late sixties. The derivation is reprinted in the book by Rechenberg (1994). The result measures the progress in x_1 direction (cf the corridor definition (B2.4.16), (B2.4.17))

$$\texttt{corridor:} \qquad \varphi_{1+1} = \frac{\sigma}{(2\pi)^{1/2}}\left[2\,\Phi_0\left(\frac{2b}{\sigma}\right) - \frac{1}{(2\pi)^{1/2}}\frac{\sigma}{b}\left(1 - \exp\left(-\frac{1}{2}\left(\frac{2b}{\sigma}\right)^2\right)\right)\right]^{n-1} \tag{B2.4.46}$$

and if $\sigma \ll b$ holds, then one obtains from equation (B2.4.46)

$$\sigma \ll b\ : \qquad \varphi_{1+1} = \frac{\sigma}{(2\pi)^{1/2}}\left[1 - \frac{1}{(2\pi)^{1/2}}\frac{\sigma}{b}\right]^{n-1}$$

and further for $n \gg 1$

$$\sigma \ll b,\ n \gg 1\ : \qquad \varphi_{1+1} = \frac{\sigma}{(2\pi)^{1/2}}\exp\left(-\frac{\sigma}{(2\pi)^{1/2}}\frac{n}{b}\right). \tag{B2.4.47}$$

As can be easily shown (from equation (B2.4.47)) φ_{1+1} has its maximum at a mutation strength $\hat{\sigma} = (2\pi)^{1/2}b/n$. The maximal progress rate $\hat{\varphi}_{1+1}$ achievable from formula (B2.4.47) is therefore $\hat{\varphi}_{1+1} = (b/n)\,\mathrm{e}^{-1}$.

The $(\tilde{1}+\tilde{1})$ result for the corridor on noisy fitness data has been derived by Rechenberg (see the reprint 1994). He obtained

$$\varphi_{\tilde{1}+\tilde{1}} = \frac{1}{\left(1 + 2(\sigma_R/\sigma)^2\right)^{1/2}}\,\varphi_{1+1} \tag{B2.4.48}$$

where σ_{R} is given by

$$\sigma_{\mathrm{R}} = \sigma_{\epsilon_Q} \Big/ \left| \frac{\mathrm{d}q_{\mathcal{C}}(x_1)}{\mathrm{d}x_1} \right|$$

and φ_{1+1} is the progress rate from the undisturbed case, equation (B2.4.47); $\mathrm{d}q_{\mathcal{C}}/\mathrm{d}x_1$ is the slope of the corridor (cf equation (B2.4.17)) at the parental state and σ_{ϵ_Q} is the absolute noise strength defined by equation (B2.4.27). As can be seen from equation (B2.4.48) noisy fitness data deteriorate the expected progress.

The $(1, \lambda)$ ES theory for the corridor has been investigated by Schwefel (reprinted in the book by Schwefel (1995)). However, at the time of writing, there is no explicit $\varphi_{1,\lambda}$ formula. Only an implicit (and approximate) $\lambda = f(\varphi, \sigma, \lambda)$ equation has been derived (see Schwefel 1995, p 139).

The second model with simple constraints is the discus, defined by equations (B2.4.18)–(B2.4.20). Results measuring the progress in the x_1 direction can be easily obtained. For the $(1 + 1)$ algorithm, for example, one obtains (Beyer 1989)

$$\texttt{discus:} \qquad \varphi_{1+1} = \frac{\sigma}{(2\pi)^{1/2}} \left[1 - \exp\left(-\frac{1}{2} \left(\frac{2b}{\sigma} \right)^2 \right) \right].$$

Maximal performance is achieved for $\hat{\sigma} \approx 1.26\, b$ which produces a progress rate $\hat{\varphi}_{1+1} \approx 0.36\, b$.

Normalization of the sphere model and smooth fitness landscapes. The exceptional role of the sphere model for the progress rate theory is threefold.

First, the model is *scalable*, i.e. by introduction of the *normalization*

$$\varphi^\star := \varphi \, \frac{n}{r} \qquad \sigma^\star := \sigma \, \frac{n}{r} \tag{B2.4.49}$$

the progress rate formulae can be expressed in such a way that they do not depend on the actual parental radius r. That is, the normalized φ becomes independent from the parental state, having a $\sigma^\star$ dependence only. This allows for easy comparison of different EAs.

Second, from the asympotical ($n \to \infty$) progress rate formulae on the sphere model one easily obtains the φ formulae for the inclined plane (see section B2.4.1.3) . This can be done by Taylor expansion of the $\varphi^\star(\sigma^\star)$ expressions breaking off after the linear $\sigma^\star$ term. Thus, one always finds a proportionality relation $\varphi(\sigma) \propto \sigma$.

Third, the sphere can serve as a local approximation for sufficiently smooth fitness landscapes which can be expanded into a Taylor series in accordance with the equations (B2.4.21), and (B2.4.22). The idea is to use the reciprocal of the local mean curvature as the (local) *mean radius* r. This is a differential geometry task. For large parameter space dimensions n one obtains (see Beyer 1995d)

$$\texttt{mean radius:} \qquad r = \frac{n}{2} \, \frac{(\boldsymbol{b}^{\mathrm{T}}\boldsymbol{b})^{1/2}}{\left| \mathrm{Tr}\{\mathbf{Q}\} - \boldsymbol{b}^{\mathrm{T}}\mathbf{Q}\boldsymbol{b}/\boldsymbol{b}^{\mathrm{T}}\boldsymbol{b} \right|} \qquad (n \gg 1) \tag{B2.4.50}$$

with $\boldsymbol{b}$ and $\mathbf{Q}$ given by equation (B2.4.22). Thus, provided that the normalized $\varphi^\star$ and $\sigma^\star$ are given for the sphere, then the local progress rate $\varphi(t)$ on the fitness landscape $F(\boldsymbol{y})$ at the point $\boldsymbol{y} = \boldsymbol{y}(t)$ can be calculated by the *renormalization equations*

$$\varphi(\sigma) = \varphi^\star_{\mathrm{sphere}}(\sigma^\star_{\mathrm{sphere}}) \frac{1}{2} \frac{(\boldsymbol{b}^{\mathrm{T}}\boldsymbol{b})^{1/2}}{\left| \mathrm{Tr}\{\mathbf{Q}\} - \boldsymbol{b}^{\mathrm{T}}\mathbf{Q}\boldsymbol{b}/\boldsymbol{b}^{\mathrm{T}}\boldsymbol{b} \right|} \quad \text{with} \quad \sigma^\star_{\mathrm{sphere}} = \sigma \, \frac{2}{(\boldsymbol{b}^{\mathrm{T}}\boldsymbol{b})^{1/2}} \left| \mathrm{Tr}\{\mathbf{Q}\} - (\boldsymbol{b}^{\mathrm{T}}\mathbf{Q}\boldsymbol{b})/(\boldsymbol{b}^{\mathrm{T}}\boldsymbol{b}) \right|. \tag{B2.4.51}$$

In the following sections $\varphi^\star$ and $\sigma^\star$ will always refer to the sphere model $\varphi^\star_{\mathrm{sphere}}$ and $\sigma^\star_{\mathrm{sphere}}$.

Note that the quality of the r approximation (B2.4.50) depends on the local fitness function $Q_{\boldsymbol{y}(t)}(\boldsymbol{x})$ (equation (B2.4.21)). It mainly depends on the eigenvalue spectrum of $\mathbf{Q}$. If this spectrum is sufficiently concentrated around $\mathrm{Tr}\{\mathbf{Q}\}/n$, then the results will be excellent. In such cases, provided that $\mathbf{Q}$ is definite, the Rayleigh quotients in equation (B2.4.51) can be dropped and the simpler formula

$$\varphi(\sigma) = \varphi^\star(\sigma^\star) \, \frac{(\boldsymbol{b}^{\mathrm{T}}\boldsymbol{b})^{1/2}}{2 \, |\mathrm{Tr}\{\mathbf{Q}\}|} \qquad \text{with} \qquad \sigma^\star = \sigma \, \frac{2 \, |\mathrm{Tr}\{\mathbf{Q}\}|}{(\boldsymbol{b}^{\mathrm{T}}\boldsymbol{b})^{1/2}}$$

yields satisfactory results.

Progress rates on ($\mu \stackrel{+}{,} \lambda$) algorithms for the $n \to \infty$ sphere. The progress rate formulae from the asymptotically ($n \to \infty$) exact theory are applicable as approximations for $\sigma^\star$ values which are not too large; as a rule of thumb

$$(\sigma^\star)^2 < n \tag{B2.4.52}$$

(see Beyer 1995b) should be fulfilled. Under this condition the (μ, λ) progress rate formula defined by equations (B2.4.11), (B2.4.12), and (B2.4.49) reads

$$\varphi^\star_{\mu,\lambda}(\sigma^\star) = c_{\mu,\lambda}\sigma^\star - \frac{(\sigma^\star)^2}{2} \tag{B2.4.53}$$

with the progress coefficient $c_{\mu,\lambda}$. (In the case $\mu > 1$, $\varphi^\star$ and $\sigma^\star$ are defined as $\varphi^\star = \varphi n/\langle r\rangle_\mu$ and $\sigma^\star = \sigma n/\langle r\rangle_\mu$, where $\langle r\rangle_\mu$ is the average radius of the μ parents.) The examples for $\mu = 1$, i.e. $c_{1,\lambda}$, have been already displayed in table B2.4.1. A collection of $\mu > 1$ progress coefficients is presented in table B2.4.2. The theory behind the $c_{\mu,\lambda}$ coefficients (for $\mu > 1$) is difficult (Beyer 1995b) and requires

Table B2.4.2. $c_{\mu,\lambda}$ progress coefficients.

μ	$\lambda = 5$	$\lambda = 10$	$\lambda = 20$	$\lambda = 30$	$\lambda = 40$	$\lambda = 50$	$\lambda = 100$	$\lambda = 150$	$\lambda = 200$	$\lambda = 250$
2	0.92	1.36	1.72	1.91	2.04	2.13	2.40	2.55	2.65	2.72
3	0.68	1.20	1.60	1.80	1.93	2.03	2.32	2.47	2.57	2.65
4	0.41	1.05	1.49	1.70	1.84	1.95	2.24	2.40	2.51	2.59
5	0.00	0.91	1.39	1.62	1.77	1.87	2.18	2.34	2.45	2.53
10	—	0.00	0.99	1.28	1.46	1.59	1.94	2.12	2.24	2.33
20	—	—	0.00	0.76	1.03	1.20	1.63	1.84	1.97	2.07
30	—	—	—	0.00	0.65	0.89	1.41	1.64	1.79	1.90
40	—	—	—	—	0.00	0.57	1.22	1.49	1.65	1.77
50	—	—	—	—	—	0.00	1.06	1.35	1.53	1.65
100	—	—	—	—	—	—	0.00	0.81	1.07	1.24

sophisticated techniques and approximation methods which are beyond the scope of this introductory article. In order to obtain a feeling of the problem to be solved the reader is reminded that the distribution of the population of μ individuals is to be determined in a *self-consistent* manner. This requires the calculation of distribution parameters derived from the (μ, λ) sampling process. The details of the cumbersome calculations can be found in the work of Beyer (1994b).

Apart from the sophisticated theory behind the $c_{\mu,\lambda}$ progress coefficients, the interpretation of the $\varphi^\star_{\mu,\lambda}$ progress rate formula in the sense of the *evolutionary progress principle* (EPP) (see section B2.4.1.1 and remark 2 in section B2.4.1.4) is straightforward. The *progress loss* is given by the $(\sigma^\star)^2/2$ term in equation (B2.4.53), whereas *progress gain* is obtained from the first term, i.e. $c_{\mu,\lambda}\sigma^\star$. The latter depends on the selection intensity quantified by the $c_{\mu,\lambda}$ coefficient. The optimal working of a (μ, λ) strategy is at $\sigma^\star = \hat{\sigma}^\star = c_{\mu,\lambda}$ because this $\sigma^\star = \hat{\sigma}^\star$ maximizes equation (B2.4.53)

$$\hat{\sigma}^\star = c_{\mu,\lambda} \qquad \Rightarrow \qquad \hat{\varphi}^\star_{\mu,\lambda} = \frac{(c_{\mu,\lambda})^2}{2}. \tag{B2.4.54}$$

As can be seen from tables B2.4.1 and B2.4.2, where maximal (local) progress is desired, the $(1, \lambda)$ version will be the best one:

$$\varphi^\star_{1,\lambda}(\sigma^\star) \geq \varphi^\star_{\mu,\lambda}(\sigma^\star). \tag{B2.4.55}$$

However, this takes only the local behavior into account, i.e. it holds exactly for the sphere model. Optimization in multimodal fitness landscapes as well as noisy fitness data may be better treated by multimembered ($\mu > 1$) ESs which allow for a certain repertoire of descendants exploring the fitness landscape.

The same argument holds for the algorithms with $(\mu + \lambda)$ elitist selection, especially used in EP (Fogel 1992). However, the analysis of the $(\mu + \lambda)$ algorithms is even more complicated than that of the

(μ, λ) ES. At the time of writing, only the $(1 + \lambda)$ algorithms can be treated (Beyer 1993):

$$\varphi^{\star}_{1+\lambda}(\sigma^{\star}) = \sigma^{\star} d^{(1)}_{1+\lambda}\left(\frac{\sigma^{\star}}{2}\right) - \frac{(\sigma^{\star})^2}{2}\left[1 - \left(\frac{1}{2} + \Phi_0\left(\frac{\sigma^{\star}}{2}\right)\right)^{\lambda}\right]. \tag{B2.4.56}$$

Here, the progress function $d^{(1)}_{1+\lambda}(x)$ is given by the integral (B2.4.43). Analytical $d^{(1)}_{1+\lambda}$ functions are known for the cases $\lambda = 1$ (equation (B2.4.44)) and $\lambda = 2$ (equation (B2.4.45)), only, and the optimal $\sigma^{\star}$ for maximal progress $\hat{\varphi}^{\star} = \varphi^{\star}(\hat{\sigma}^{\star})$ can be obtained by numerical techniques only. However, if λ decreases, $\hat{\sigma}^{\star}$ asymptotically approaches the $\hat{\sigma}^{\star}$ value of the $(1, \lambda)$ ES. As a rule of thumb $\lambda \overset{>}{\approx} 20 : \hat{\sigma}^{\star} \to c_{1,\lambda}$ holds.

Like the inequality (B2.4.55) which holds for $(\,,\,)$ strategies, a similar inequality can be formulated for the $(+)$ algorithms:

$$\varphi^{\star}_{1+\lambda}(\sigma^{\star}) \geq \varphi^{\star}_{\mu+\lambda}(\sigma^{\star}).$$

This is plausible because the $\mu > 1$ selection allows the procreation of offspring from worse parents, whereas the $\mu = 1$ selection always takes the best.

Because of the elitist $(+)$ selection the progress rate in $(\mu+\lambda)$ algorithms on the sphere model cannot be negative, it always holds that

$$\varphi^{\star}_{\mu+\lambda}(\sigma^{\star}) \geq 0. \tag{B2.4.57}$$

This is in contrast to the $(\,,\,)$ strategies which can have negative progress rates, due to the unbounded loss part in equation (B2.4.53).

Generally, $(\mu + \lambda)$ algorithms can exhibit slightly larger progress rates. For the case $\mu = 1$ one can derive the inequality

$$\varphi^{\star}_{1+\lambda}(\sigma^{\star}) \geq \varphi^{\star}_{1,\lambda}(\sigma^{\star})$$

from the associated progress rate integral (see Beyer 1993, p 171). It is conjectured that this relation can be extended to $\mu > 1$:

$$\texttt{conjecture:} \qquad \varphi^{\star}_{\mu+\lambda}(\sigma^{\star}) \geq \varphi^{\star}_{\mu,\lambda}(\sigma^{\star}).$$

A proof for this inequality, apart from the trivial case $\mu = \lambda$, is still pending.

Multirecombinant intermediate $(\mu/\mu_{\mathrm{i}}, \lambda)$ evolution strategies. In general, the analysis of EAs with recombination is much more difficult than the simple mutation–selection algorithms. However, there are some exceptions on $(\mu/\rho, \lambda)$ ESs using *multirecombination* (see Section C3.3.2 for the definition) which C3.3.2
are both relatively simple to analyze and very powerful concerning their local performance (progress rate) properties. There are two variants.

First, the $\rho = \mu_{\mathrm{i}}$ *intermediate recombination* performs the multimixing in terms of a *center of mass* recombination, i.e. the μ selected parents are averaged in accordance with equation (B2.4.13). After that, the mutations are applied to the *center of mass parent* to create λ new offspring.

Second, the $\rho = \mu_{\mathrm{d}}$ *dominant recombination*, often called *global discrete recombination*, recombines the μ parents coordinatewise in order to produce one offspring (which is mutated after that) by choosing randomly one of the μ coordinate values (i.e. *dominant* in contrast to the averaging in $\rho = \mu_{\mathrm{i}}$ algorithms). This strategy will be treated in the next section.

Because of the high progress rates obtained for relatively large $\sigma^{\star}$ values, the condition (B2.4.52) for the applicability of the $n \to \infty$ asymptotical theory is more or less violated. Therefore, the n dependence must be taken into account. This has been done by Beyer (1995c). For the intermediate $\rho = \mu_{\mathrm{i}}$ ES one obtains

$$\varphi^{\star}_{\mu/\mu_{\mathrm{i}},\lambda}(\sigma^{\star}) = \sigma^{\star} c_{\mu/\mu,\lambda} \frac{1 + (\sigma^{\star})^2/2\mu n}{(1 + (\sigma^{\star})^2/2n)^{1/2}(1 + (\sigma^{\star})^2/\mu n)^{1/2}} + n\left(1 - \left(1 + \frac{(\sigma^{\star})^2}{\mu n}\right)^{1/2}\right) \tag{B2.4.58}$$

with the $c_{\mu/\mu,\lambda}$ progress coefficient

$$c_{\mu/\mu,\lambda} = \frac{\lambda - \mu}{2\pi}\binom{\lambda}{\mu}\int_{-\infty}^{\infty} \mathrm{e}^{-t^2}\left(\frac{1}{2} + \Phi_0(t)\right)^{\lambda-\mu-1}\left(\frac{1}{2} - \Phi_0(t)\right)^{\mu-1} \mathrm{d}t$$

Table B2.4.3. The $c_{\mu/\mu,\lambda}$ progress coefficients.

μ	$\lambda = 10$	$\lambda = 20$	$\lambda = 30$	$\lambda = 40$	$\lambda = 50$	$\lambda = 100$	$\lambda = 150$	$\lambda = 200$
2	1.270	1.638	1.829	1.957	2.052	2.328	2.478	2.580
3	1.065	1.469	1.674	1.810	1.911	2.201	2.357	2.463
4	0.893	1.332	1.550	1.694	1.799	2.101	2.263	2.372
5	0.739	1.214	1.446	1.596	1.705	2.018	2.185	2.297
10	0.000	0.768	1.061	1.242	1.372	1.730	1.916	2.040
20	—	0.000	0.530	0.782	0.950	1.386	1.601	1.742
30	—	—	0.000	0.414	0.634	1.149	1.390	1.545
40	—	—	—	0.000	0.343	0.958	1.225	1.393
50	—	—	—	—	0.000	0.792	1.085	1.265
100	—	—	—	—	—	0.000	0.542	0.795

tabulated in table B2.4.3. Note that equation (B2.4.58) includes the n-dependent case $\rho = \mu = 1$, i.e. the n-dependent $(1, \lambda)$ ES where $c_{1,\lambda} = c_{1/1,\lambda}$ holds. The $c_{\mu/\mu,\lambda}$ coefficients obey the relation

$$0 \leq c_{\mu/\mu,\lambda} \leq c_{\mu,\lambda} \leq c_{1,\lambda}. \tag{B2.4.59}$$

They are asymptotically equal to $(\lambda \to \infty)$

$$c_{\mu/\mu,\lambda} \sim \frac{\lambda}{\mu} \frac{1}{(2\pi)^{1/2}} \exp\left[-\frac{1}{2}\left(\Phi_0^{-1}\left(\frac{1}{2} - \frac{\mu}{\lambda}\right)\right)^2\right] \qquad 0 < \frac{\mu}{\lambda} < 1. \tag{B2.4.60}$$

(for $\Phi_0^{-1}(x)$ see the remark on equation (B2.4.42).) Equation (B2.4.60) can serve as an approximate $c_{\mu/\mu,\lambda}$ formula for $\lambda \gtrapprox 1000$. By asymptotical iteration one finds from equation (B2.4.60) that $c_{\mu/\mu,\lambda}$ is of order

$$c_{\mu/\mu,\lambda} = \mathrm{O}\left(\left(\ln\left(\frac{\lambda}{\mu}\right)\right)^{1/2}\right).$$

In order to see the main difference between (μ, λ) and $(\mu/\mu, \lambda)$ ESs it is worth investigating the asymptotic $n \to \infty$ case which can be obtained by Taylor expansion of the square roots for $(\sigma^\star)^2/n \ll 1$ in equation (B2.4.58)

$$n \to \infty: \qquad \varphi^\star_{\mu/\mu_\mathrm{i},\lambda}(\sigma^\star) = \sigma^\star c_{\mu/\mu,\lambda} - \frac{1}{\mu}\frac{(\sigma^\star)^2}{2} \tag{B2.4.61}$$

(see also Rechenberg (1994)).

Let us investigate this result in the light of the EPP hypothesis. As can be seen, the *progress loss* in equation (B2.4.61) is smaller by a factor of $1/\mu$ than that from the (μ, λ) ES (equation (B2.4.53)). Although the *progress gain* is not increased (assuming equal $\sigma^\star$ values), because of inequality (B2.4.59), $c_{\mu/\mu,\lambda} \leq c_{\mu,\lambda}$. The maximal achievable progress rate becomes

$$\hat{\varphi}^\star_{\mu/\mu,\lambda}(\sigma^\star) = \mu \frac{c^2_{\mu/\mu,\lambda}}{2} \qquad \text{at} \qquad \hat{\sigma}^\star = \mu\, c_{\mu/\mu,\lambda}. \tag{B2.4.62}$$

The main effect of recombination is the *reduction* of the progress loss. This allows for larger mutation strengths $\hat{\sigma}^\star$ with the result of a larger progress rate. The deeper reason for this behavior can be traced back to the intermediate averaging acting on the mutations $\boldsymbol{z}_{m;\lambda}$ of the μ best offspring. These $\boldsymbol{z}_{m;\lambda}$ can be decomposed into a component x in optimum direction $\boldsymbol{e}_\mathrm{opt}$ (unit vector) and a perpendicular part $\boldsymbol{h}$

$$\boldsymbol{z}_{m;\lambda} = x_{m;\lambda}\boldsymbol{e}_\mathrm{opt} + \boldsymbol{h}_{m;\lambda} \qquad \boldsymbol{e}^\mathrm{T}_\mathrm{opt}\boldsymbol{h}_{m;\lambda} = 0.$$

The directions of the $\boldsymbol{h}_{m;\lambda}$ parts are selectively neutral, whereas the $x_{m;\lambda}$ are selected according to their length (optimum direction). By this decomposition it becomes clear that the $\boldsymbol{h}$ vectors are the *harmful* components of the mutations. Now, perform the intermediate recombination, i.e. the averaging (B2.4.13).

Because the different mutation vectors $\boldsymbol{z}_\lambda$ are statistically independent of each other and so are the $\boldsymbol{h}_{m;\lambda}$ vectors, the averaging of these $\boldsymbol{h}_{m;\lambda}$ vectors produces a recombinative $\boldsymbol{h}$ vector $\langle \boldsymbol{h} \rangle_\mu$ with an expected length square $\|\langle \boldsymbol{h} \rangle_\mu\|^2$ which is smaller by a factor of $1/\mu$ than the expected length square of a single $\boldsymbol{h}$. Thus, the harmful part of the mutations is decreased by a factor of roughly $1/\mu$. (This is the reason why the $(\mu/\rho, \lambda)$ ESs with $\rho = \mu$ can exhibit larger (local) progress rates than the 'bisexual' ($\rho = 2$) variant.) Note that the averaging of the $x_{m;\lambda}$ values *does not* produce a larger x component, because $\langle x \rangle_\mu < x_{\lambda:\lambda}$ always holds. That is, the recombination *does not* produce better descendants by superposition of good 'partial solutions' (as conjectured by the building block hypothesis, Section B2.5.3). It rather performs a B2.5.3 *statistical error correction* that diminishes the influence of the harmful part of the mutations. This effect of error correction has been termed *genetic repair* (GR) (Beyer 1995c). The GR hypothesis builds up an alternative explanation for the working of recombination in EAs diametrically opposed to the building block hypothesis (Goldberg 1989).

The limit $n \to \infty$ case (equations (B2.4.61), and (B2.4.62)) is well suited to discussing the GR principle and the benefits of multirecombination; however, it considerably deviates quantitatively from the real-world case $n < \infty$ (equation (B2.4.58)). This becomes especially evident if one asks for the optimal number $\hat{\mu}$ of parents for a given (fixed) number λ of offspring. The asymptotical theory yields $\hat{\mu} \approx 0.27\,\lambda$ which can be obtained by maximization of the $\hat{\varphi}^\star$ formula (B2.4.62) with respect to μ (for fixed λ). The numerical analysis of equation (B2.4.58), however, reveals a relatively strong dependence of the optimal number $\hat{\mu}$ of parents on the parameter space dimension n.

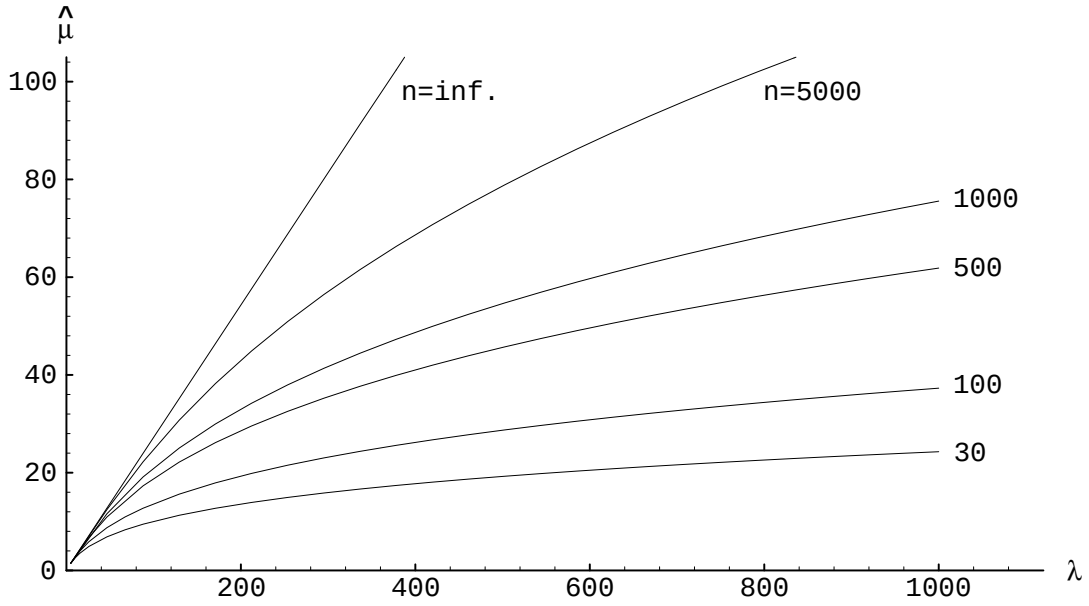


Figure B2.4.1. The optimal number $\hat{\mu}$ of parents in $(\mu/\mu, \lambda)$ ESs depends on the number of offspring λ and on the parameter space dimension n.

Figure B2.4.1 displays the results of the numerical analysis. For example, for $\lambda = 100$ one finds $\hat{\mu}(n = \infty) = 27$, but $\hat{\mu}(n = 30) = 10$; for $\lambda = 1000$, $\hat{\mu}(n = \infty) = 270$ is predicted, but for $n = 100$ one finds $\hat{\mu} = 37$. An extensive table of the optimal μ choice can be found in the article by Beyer (1995c).

Multirecombinant dominant $(\mu/\mu_d, \lambda)$ evolution strategies. The theoretical analysis of the global discrete recombination pattern (dominant recombination) is still in the early stages. The results obtained provide rough estimates for the progress rate. The main problem of the analysis is concentrated on the determination of the parental distribution. A first approximation to this problem is given by the assumption of isotropic, normally distributed *surrogate mutations* $\boldsymbol{s}$ generated from an *imaginary center of mass parent.* Due to the isotropy assumption the examination of a single vector component of $\boldsymbol{s}$ suffices, and the determination of the parental distribution reduces to the calculation of the standard deviation σ_s of the surrogate mutations.

Given the (physical) mutation distribution equation (B2.4.28), these mutations are 'transformed' by the repeated (i.e. over the generations) recombination process into the surrogate mutations. Their steady-state standard deviation exhibits a saturation behavior; one finds

$$\sigma_{\mathrm{s}} = \mu^{1/2}\,\sigma$$

(Beyer 1995c). That is, starting from a parental population concentrated at *one* point ($\sigma_{\mathrm{s}} = 0$), the process 'dominant recombination' *and* '(physical) mutations with σ' *iteratively* performed (over the generations) produces a parental distribution (with $\sigma_{\mathrm{s}} \neq 0$) which is larger by a factor of $\mu^{1/2}$ than the generating (i.e. physical) mutations. The result of this process looks just like a *speciation* in biology. The individuals are crowded around an (imaginary) *wild type*. A very interesting aspect of this result is that the σ_{s} approaches a steady-state value, i.e. σ_{s} is restricted for $t \to \infty$. Note that this result has been obtained *without* selection (there is no selective pressure in the surrogate mutation model). Because this result is important for all discrete/dominant recombination schemes it is regarded as a basic principle of EAs—the *mutation-induced speciation by recombination* principle, MISR for short.

As already pointed out, the isotropy assumption is the weak point in the model. Due to the selection the 'shape' of the surrogate mutation density will be distorted. Therefore one cannot expect quantitatively exact progress rate formulae for the real-world case ($n < \infty$). This holds for

$$\varphi^{\star}_{\mu/\mu_{\mathrm{d}},\lambda}(\sigma^{\star}) = \frac{\mu^{1/2}\,\sigma^{\star} c_{\mu/\mu,\lambda}}{(1+(\sigma^{\star})^2/n)^{1/2}} \;+\; n\left(1-\left(1+\frac{(\sigma^{\star})^2}{n}\right)^{1/2}\right) \tag{B2.4.63}$$

(Beyer 1995c), which yields satisfactory results for $n > 1000$, as well as the asymptotic case

$$n \to \infty: \qquad \varphi^{\star}_{\mu/\mu_{\mathrm{d}},\lambda}(\sigma^{\star}) = \mu^{1/2}\,\sigma^{\star} c_{\mu/\mu,\lambda} \;-\; \frac{(\sigma^{\star})^2}{2}$$

(Rechenberg 1994) which can be easily derived from equation (B2.4.63) for $(\sigma^{\star})^2/n \ll 1$.

Progress rates for ($\tilde{1}\,\dot{+}\,\tilde{\lambda}$) algorithms on noisy fitness data. Noisy fitness data always degrade the EA performance as has been seen, for example, in the case of the corridor (equation (B2.4.48)).

For the spherical model the asymptotical theory $n \to \infty$ yields in the case of the $(\tilde{1},\tilde{\lambda})$ ES (Beyer 1993)

$$\varphi^{\star}_{\tilde{1},\tilde{\lambda}}(\sigma^{\star}) = c_{1,\lambda}\sigma^{\star}\frac{1}{(1+(\sigma^{\star}_{\epsilon}/\sigma^{\star})^2)^{1/2}} \;-\; \frac{(\sigma^{\star})^2}{2} \tag{B2.4.64}$$

where $\sigma^{\star}_{\epsilon}$ is the normalized noise strength

$$\sigma^{\star}_{\epsilon} = \sigma_{\epsilon_Q}\,\frac{n}{r\,Q'} \qquad \text{with} \qquad Q' := \left\| \nabla Q_{\boldsymbol{y}(t)}(\boldsymbol{x})\big|_{\boldsymbol{x}=\boldsymbol{o}} \right\| \tag{B2.4.65}$$

σ_{ϵ_Q} is the standard deviation of the measuring error (cf equations (B2.4.26), and (B2.4.27)) and Q' is the absolute value of the normal derivative of the local quality function Q at $\boldsymbol{x} = \boldsymbol{0}$. For example, if the local quality function depends on the radius of the sphere model such that $Q(r) = \hat{Q} - cr^2$ holds ($\hat{Q}$ = constant), then $\sigma^{\star}_{\epsilon}$ is obtained from equation (B2.4.65) as follows:

$$Q_{\boldsymbol{y}(t)}(\boldsymbol{x}) = Q_{\boldsymbol{y}(t)}(r) = \hat{Q} - cr^2 \qquad \Rightarrow \qquad \sigma^{\star}_{\epsilon} = \sigma_{\epsilon_Q}\frac{n}{2cr^2}.$$

The influence of the noise on the progress rate can be studied by comparison with the undisturbed case, equation (B2.4.53), for $\mu = 1$. Again, the *evolutionary progress principle* (EPP) is observed. The noise decreases the progress gain. This is because the fitness noise 'deceives' the selection process. The progress loss, however, is not changed. The 'deception' can be so strong that even an evolutionary progress is excluded. The necessary evolution condition is $\varphi^{\star} > 0$. If applied to equation (B2.4.64) one easily finds

$$(\tilde{1},\tilde{\lambda})\ \texttt{ES necessary evolution condition:} \qquad \sigma^{\star}_{\epsilon} < 2c_{1,\lambda}.$$

That is, if $\sigma^{\star}_{\epsilon}$ is larger than $2c_{1,\lambda}$, then the $(\tilde{1},\tilde{\lambda})$ ES cannot converge at all.

The $(\tilde{1}+\tilde{\lambda})$ algorithms do not have a generally closed analytical expression for the progress rate $\varphi^{\star}_{\tilde{1}+\tilde{\lambda}}$. So far, only the $\lambda = 1$ case can be treated analytically:

$$\varphi^{\star}_{\tilde{1}+\tilde{1}}(\sigma^{\star}) = \sigma^{\star}\,\frac{1}{(1+2(\sigma^{\star}_{\epsilon}/\sigma^{\star})^2)^{1/2}}\,d^{(1)}_{1+1}\left(\frac{\sigma^{\star}}{2}\,\frac{1}{(1+2(\sigma^{\star}_{\epsilon}/\sigma^{\star})^2)^{1/2}}\right) - \frac{(\sigma^{\star})^2}{2}\left[\frac{1}{2}-\Phi_0\left(\frac{\sigma^{\star}}{2}\,\frac{1}{(1+2(\sigma^{\star}_{\epsilon}/\sigma^{\star})^2)^{1/2}}\right)\right].$$

The comparison with equation (B2.4.56) ($\lambda = 1$) shows that—unlike the $(\tilde{1},\tilde{\lambda})$ ESs—in $(+)$ selection (elitist) algorithms both the progress gain *and* the progress loss are affected by the noise. This also holds for algorithms with $\lambda \geq 2$. For these algorithms, however, there exists only an integral representation (see Beyer 1993):

$$\begin{aligned}\varphi^{\star}_{\tilde{1}+\tilde{\lambda}}(\sigma^{\star}) = {} & \frac{\sigma^{\star}}{(1+2(\sigma^{\star}_{\epsilon}/\sigma^{\star})^2)^{1/2}}\,\frac{\lambda}{(2\pi)^{1/2}}\int_{-\infty}^{\infty} t\,\mathrm{e}^{-\frac{1}{2}t^2}\left[\frac{1}{2}+\Phi_0(t)\right]^{\lambda-1} \\ & \times\left[\frac{1}{2}+\Phi_0\left(\left(1+\left(\frac{\sigma^{\star}}{\sigma^{\star}_{\epsilon}}\right)^2\right)^{1/2}t-\frac{(\sigma^{\star})^2}{2\sigma^{\star}_{\epsilon}}\right)\right]\mathrm{d}t \\ & -\frac{(\sigma^{\star})^2}{2}\left\{1-\left(1+\left(\frac{\sigma^{\star}}{\sigma^{\star}_{\epsilon}}\right)^2\right)^{1/2}\frac{1}{(2\pi)^{1/2}}\int_{-\infty}^{\infty}\left[\frac{1}{2}+\Phi_0(t)\right]^{\lambda}\right. \\ & \left.\times\exp-\frac{1}{2}\left(\left(1+\left(\frac{\sigma^{\star}}{\sigma^{\star}_{\epsilon}}\right)^2\right)^{1/2}t-\frac{(\sigma^{\star})^2}{2\sigma^{\star}_{\epsilon}}\right)^2\mathrm{d}t\right\}.\end{aligned}$$

An astonishing observation in $(+)$ selection algorithms on noisy fitness landscapes is the nondefiniteness of the progress rate φ which is opposite to the standard where $\varphi_{\mu+\lambda} \geq 0$, inequality (B2.4.57), is always ensured. The $\varphi_{\mu+\lambda} \geq 0$ property guarantees the convergence of the algorithm. In noisy fitness landscapes, however, $\varphi_{\tilde{\mu}+\tilde{\lambda}}$ depends on the noise strength $\sigma^{\star}_{\epsilon}$. If $\sigma^{\star}_{\epsilon}$ is too large, then $\varphi_{\tilde{\mu}+\tilde{\lambda}} < 0$ and the algorithm cannot converge. *As can be seen, elitism does not always guarantee convergence.* There are further methods/principles which should be considered, too, in order to improve the convergence security, such as the *democracy principle* (Beyer 1993, p 186) and the *multiparent* (i.e. population-based) strategies (Rechenberg 1994, p 228).

Theoretical results on multiparent (μ,λ) or $(\mu/\mu,\lambda)$ ESs are not available at the time of writing. There is some strong empirical evidence that such strategies are much more advantageous in noisy fitness landscapes than the $(1\,\dot{+}\,\lambda)$ ESs (see Rechenberg 1994).

B2.4.1.6 Dynamics and convergence order of evolutionary algorithms

General aspects. Unlike the progress rate describing the microscopic change of the population from generation t to $t+1$, the dynamics refers to the *macroscopic* aspects of the EA. Topics of the evolution dynamics are first of all the questions of whether there is convergence and how fast the EA approaches the optimum (see also Section B2.2 and B2.3). B2.2, B2.3

The latter requires the time evolution of the residual distance $h(t)$ to the optimum (cf equation (B2.4.9)). Let $r(t) = \mathrm{E}\{h(t)\}$ be the expectation of the distance, then by virtue of definition (B2.4.8) one has

$$r(t+1) = r(t) - \varphi(t).$$

This difference equation can be approximated by a differential equation. If, for example, the model class hypersphere is considered, r becomes the (local average) radius and with the normalization (B2.4.49) one finds

$$r(t+1) - r(t) = -r(t)\,\frac{1}{n}\varphi^{\star}(\sigma^{\star}(t)).$$

Taylor expansion of $r(t+1)$ yields $r(t+1) = r(t) + (\mathrm{d}r/\mathrm{d}t)\,1 + \ldots$; thus, one obtains the differential equation of the r evolution

$$\frac{\mathrm{d}r(t)}{\mathrm{d}t} = -r(t)\,\frac{1}{n}\varphi^{\star}(\sigma^{\star}(t)). \tag{B2.4.66}$$

Note that this approximation is possible because of the smallness of $\varphi^{\star}$, i.e. $\varphi^{\star}/n \ll 1$.

Equation (B2.4.66) can be formally solved for $r(t)$. Given a initial distance $r(0)$ at the start of the EA run the solution reads

$$r(t) = r(0)\exp\left\{-\frac{1}{n}\int_{t'=0}^{t'=t}\varphi^{\star}(\sigma^{\star}(t'))\,\mathrm{d}t'\right\}. \tag{B2.4.67}$$

As can be seen, $r(t)$ depends on the time evolution of the normalized mutation strength $\sigma^{\star}$. The special case $\sigma^{\star} =$ constant will be discussed in the next section, whereas the next but one is devoted to the $\sigma =$ constant case.

Evolutionary dynamics for $\sigma^{\star} = constant$. Maximal (local) EA performance is achieved for that normalized mutation strength $\sigma^{\star} = \hat{\sigma}^{\star}$ which maximizes the progress rate $\varphi^{\star}$. Therefore, the assumption $\sigma^{\star} =$ constant $\approx \hat{\sigma}^{\star}$ is a desired working regime which is usually attained by special σ control rules. These rules change the mutation strength σ in such a way that it can keep pace with the r change such that $\sigma(t)n/r(t) = \sigma^{\star} \approx$ constant is roughly fulfilled. The most prominent algorithm that achieves this tuning in a very natural, i.e. evolutionary, fashion, is the *self-adaptation* ES developed by Rechenberg and Schwefel in the late sixties (see e.g. Schwefel 1995) which is also widely used in EP (Fogel 1992, 1995). C7.1

The analysis of self-adaptation on $(1,\lambda)$ ESs can be found in the article by Beyer (1996). It will be assumed here that the σ control mechanism works such that $\sigma^{\star} \approx$ constant and furthermore $\varphi^{\star}(\sigma^{\star}) > 0$ is fulfilled. Then, from equation (B2.4.67) one finds the exponential time law

$$r(t) = r(0)\,\exp\left(-\frac{\varphi^{\star}(\sigma^{\star})}{n}\,t\right). \tag{B2.4.68}$$

That is, the EA approaches the optimum exponentially. Such behavior is also known as *linear convergence order*. This notion becomes clear if one switches to the logarithmic scale,

$$\ln(r(t)) = \ln(r(0)) - \frac{\varphi^{\star}(\sigma^{\star})}{n}\,t.$$

On the logarithmic r scale the time evolution becomes a linear function of the generation number t. An astonishing observation in practice is that well-designed EP and ES algorithms do exhibit this time behavior even for multimodal real-valued optimization problems. It is conjectured that this is because of the curvature properties in higher-dimensional fitness landscapes, which can be well approximated by the mean curvature equivalent to a (local) hypersphere.

Evolutionary dynamics of $(,)$ strategies for $\sigma = constant$. Let us assume that for some reason the σ control does not work, i.e. $\sigma^{\star} \neq$ constant, and the mutation strength σ remains constant. Due to equation (B2.4.49), $\sigma^{\star} = \sigma n/r(t)$, the normalized mutation strength $\sigma^{\star}$ becomes a monotonically increasing function of t, provided that r decreases with t. In $(+)$ selection algorithms (without fitness noise, see section B2.4.1.5) this simply degrades the progress rate $\varphi^{\star}$; convergence is still guaranteed (see also Fogel 1994), but with a *sublinear* convergence order. However, this does *not* hold for $(,)$ ESs. These strategies exhibit an r saturation, i.e. a steady-state r_{∞} value is reached for $t \to \infty$. The steady-state solution can be easily obtained from the differential equation (B2.4.66) by the substitution $\sigma^{\star} = \sigma n/r$ and the steady-state condition $\mathrm{d}r/\mathrm{d}t = 0$

$$\texttt{steady-state:} \qquad \sigma = \text{constant} > 0 \quad \wedge \quad \frac{\mathrm{d}r}{\mathrm{d}t} = 0 \quad \Leftrightarrow \quad \varphi^{\star}(\sigma^{\star}) = 0 \quad \Leftrightarrow \quad \sigma^{\star} = \sigma_0^{\star} > 0.$$

Here, $\sigma_0^{\star}$ is the (second) zero of $\varphi^{\star}$ (NB, the first zero is $\sigma^{\star} = 0$). If renormalized one obtains the steady-state value

$$r_{\infty} = \frac{\sigma n}{\sigma_0^{\star}} > 0. \tag{B2.4.69}$$

That is, for $t \to \infty$ a residual distance to the optimum point remains. In other words, $(\,,\,)$ strategies ***without*** σ *control are not function optimizers*; they do not converge to the optimum.

For example, consider the (μ, λ) ES. From equation (B2.4.53) one obtains

$$(\mu, \lambda) \text{ ES, } \sigma = \text{constant} > 0: \qquad \sigma_0^\star = 2c_{\mu,\lambda} \qquad \Rightarrow \qquad r_\infty = \frac{\sigma n}{2c_{\mu,\lambda}}. \tag{B2.4.70}$$

Similar results can be obtained (or observed in simulations) for all $(\,,\,)$ strategies including bitstring optimizations (for $p_\mathrm{m} = \text{constant} > 0$, see e.g. the `OneMax` $\varphi_{1,\lambda} = \bar{Q}_{1,\lambda}$ (B2.4.41)) and combinatorial problems (e.g. ordering problems; see Section G4.2). G4.2

B2.4.2 Genetic algorithms

Günter Rudolph

Abstract

The expectation of the random time at which a genetic algorithm (GA) detects the global solution or some other element of a distinguished set for the first time represents a useful global performance measure for the GA. In this section it is shown how to deduce bounds on the global performance measure from local performance measures in the case of GAs with elitist selection, mutation, and crossover.

B2.4.2.1 Global performance measures

Let the tuple $P_t = (\boldsymbol{X}_t^{(1)}, \ldots, \boldsymbol{X}_t^{(\mu)})$ denote the random population of $\mu < \infty$ individuals at generation $t \geq 0$ with $\boldsymbol{X}_t^{(i)} \in \mathbb{B}^\ell = \{0,1\}^\ell$ for $i = 1, \ldots, \mu$. Assume that the genetic algorithm (GA) is used to find a global solution $\boldsymbol{x}^* \in \mathbb{B}^\ell$ at which the objective function $f : \mathbb{B}^\ell \to \mathbb{R}$ attains its global maximum $f(\boldsymbol{x}^*) = f^* = \max\{f(\boldsymbol{x}) : \boldsymbol{x} \in \mathbb{B}^\ell\}$. The best objective function value of a population P_t at generation $t \geq 0$ can be extracted via the mapping

$$f_\mathrm{b}(P_t) = \max\{f(\boldsymbol{X}_t^{(i)}) : i = 1, \ldots, \mu\}.$$

Then the random variable

$$T = \min\{t \geq 0 : f_\mathrm{b}(P_t) = f^*\}$$

denotes the *first hitting time* of the GA. Assume that the expectation of T can be bounded via $\mathbf{E}[\,T\,] \leq \widehat{T}$ B2.2.2 where $\widehat{T}$ is a polynomial in the problem dimension ℓ. If the GA is stopped after $c\,\widehat{T}$ steps with $c \geq 2$ one cannot be sure in general whether the best solution found is the global solution or not. The probability that the candidate solution is not the global solution is $\mathbf{P}\{T > c\,\widehat{T}\}$ which can be bounded via the Markov inequality yielding

$$\mathbf{P}\{T > c\,\widehat{T}\} \;\leq\; \frac{\mathbf{E}[\,T\,]}{c\,\widehat{T}} \;\leq\; \frac{\widehat{T}}{c\,\widehat{T}} \;=\; \frac{1}{c} \;\leq\; \frac{1}{2}.$$

After k independent runs (with different random seeds) the probability that the global solution has been found at least once is larger than or equal to $1 - c^{-k}$. For example, after 20 runs with $c = 2$ (possibly in parallel) the probability that the global solution has not been found is less than 10^{-6}.

If such a polynomial bound $\widehat{T}$ existed for some evolutionary algorithm and a class of optimization problems whose associated decision problem is nondeterministic polynomial-time (NP) complete, every optimization problem of this difficulty could be treated with this fictitious evolutionary algorithm in a similar manner. In fact, for the field of evolutionary algorithms this would be a pleasant result, but such a result is quite unlikely to hold.

B2.4.2.2 Deducing the global performance measure from the local performance measure

The moments of the first hitting time can be calculated from the transition matrix of the *Markov chain* B2.2.2 associated with the GA and the objective function under consideration. Unless the transition matrix is sparsely filled the practical application of the formulas given by Iosifescu (1980, pp 104, 133) is usually excluded.

The general idea to circumvent this problem is as follows. Let $S = \mathbb{B}^\ell \times \cdots \times \mathbb{B}^\ell = (\mathbb{B}^\ell)^\mu$ be the *state space* of the Markov chain. Then each element $\boldsymbol{x} \in S$ represents a potential population that can be attained by the GA in the course of the search. Notice that it is always possible to decompose the state space into n subsets via B2.2.2

$$S = \bigcup_{i=1}^{n} S_i \quad \text{with} \quad S_i \cap S_j = \emptyset \quad \text{for} \quad i \neq j$$

with the property

$$\forall \boldsymbol{x} \in S_i : \forall y \in S_j : i < j \Rightarrow f_{\mathrm{b}}(\boldsymbol{x}) < f_{\mathrm{b}}(\boldsymbol{y}).$$

If the GA employs *elitist selection* it is guaranteed that a population in subset S_j will never transition to a population represented by a state in some subset S_i with $i < j$. Thus, the Markov chain moves through the sets S_i with ascending index i. In general, this grouping of the states does not constitute a Markov chain whose states are represented by the sets S_i (see Iosifescu 1980, pp 166–70). In this case one has to determine a lower bound on the probabilities to transition from some arbitrary element in S_i to an arbitrary element in S_j. These lower bounded probabilities represent the transition probabilities p_{ij} for the grouped Markov chain to transition from set S_i to S_j. After the probabilities p_{ij} have been determined for $j = i+1, \ldots, n$ the setting C2.7.4

$$p_{ii} = 1 - \sum_{j=i+1}^{n} p_{ij}$$

ensures that the row sums of the transition matrix of the grouped Markov chain are unity.

If the mutation of an individual is realized by inverting each bit with some *mutation probability* $p \in (0, 1)$ then there exist nonzero transition probabilities to move from set S_i to S_j for all indices i, j with $1 \leq i < j \leq n$. This Markov chain can be simplified by setting C3.2.1

$$\begin{aligned} q_{ii} &= p_{ii} + \sum_{j=i+2}^{n} p_{ij} \\ q_{i,i+1} &= p_{i,i+1} \\ q_{i,i+k} &= 0 \qquad \text{for } k \geq 2 \text{ and } i + k \leq n \\ q_{ij} &= 0 \qquad \text{for } j < i. \end{aligned}$$

Thus, only transitions from the set S_i to S_{i+1} for $i = 1, \ldots, n-1$ are considered—the remaining improving transitions are ignored by bending them back to state S_i. Evidently, this simplified Markov chain must have a worse performance than the original Markov chain, but its simple structure allows an easy determination of the first hitting time representing an upper bound on the first hitting time of the original chain. To this end, let T_{ij} denote the random time that is necessary to transition from set S_i to S_j. Then the expectation of T is bounded by

$$\mathbf{E}[\, T \,] \leq \sum_{i=1}^{n-1} \mathbf{E}[\, T_{i,i+1} \,]. \tag{B2.4.71}$$

Evidently, the probability distribution of random variable $T_{i,i+1}$ is geometric with probability density function

$$\mathbf{P}\{T_{i,i+1} = \tau\} = q_{i,i+1}\,(1 - q_{i,i+1})^{\tau - 1}$$

and expectation $\mathbf{E}[\, T_{i,i+1} \,] = 1/q_{i,i+1}$. Consequently, the expectation of the first hitting time T of the GA can bounded by

$$\mathbf{E}[\, T \,] \leq \sum_{i=1}^{n-1} \frac{1}{q_{i,i+1}}. \tag{B2.4.72}$$

It is not guaranteed that this approach will always lead to sharp bounds. The manner in which the state space is decomposed determines the quality of the bounds. Unfortunately, there is currently no guideline helping to decide which partitioning will be appropriate. The following examples will offer the opportunity to gain some experience, but before beginning the examples notice that it can be sufficient to analyze the $(1+1)$ GA with mutation and elitist selection to obtain an upper bound of the first hitting time: an ordinary GA with elitist selection, mutation, and crossover is at least as fast as a $(1+1)$ GA with the same mutation probability. Thus, the potential improving effects of crossover will be ignored. This can lead to weak bounds—but as long as the bounds are polynomially bounded in ℓ this approach is reasonable.

B2.4.2.3 Linear binary problems

Definition B2.4.1. A function $f : \mathbb{B}^\ell \to \mathbb{R}$ is called *linear* if it is representable via

$$f(\boldsymbol{x}) = a_0 + \sum_{i=1}^{\ell} a_i \, x_i$$

with $\boldsymbol{x} \in \mathbb{B}^\ell$ and $a_i \in \mathbb{R}$ for $i = 0, 1, \ldots, \ell$. □

The so-called *counting ones problem* consists of the task of finding the maximum of the linear function

$$f(\boldsymbol{x}) = \sum_{i=1}^{\ell} x_i$$

that is attained if all entries in vector $\boldsymbol{x}$ are set to 1. Bäck (1992) investigated this problem for a $(1+1)$ GA with mutations as described previously and derived the transition probabilities for the Markov chain while Mühlenbein (1992) succeeded in calculating an approximation of the expected number of function evaluations needed to reach the optimum.

The first step of the analysis is to reduce the state space of the Markov chain by an appropriate grouping of states: to this end note that there are $\binom{\ell}{i}$ states with i ones that can be grouped into one state because the specific instantiation of the vector with exactly i ones is not important—the probability of transition to any other state only depends on the number of ones (or zeros). Thus, the states of the grouped Markov chain represent the number of ones. This reduces the cardinality of the state space from 2^ℓ to $\ell + 1$: the Markov chain is in state $i \in \{0, 1, \ldots, \ell\}$ if there are exactly i ones in the current vector. Consequently, one would like to know the expected time to reach state ℓ.

The next step consists of the determination of the transition probabilities. Since the algorithm only accepts improvements it is sufficient to know the transition probabilities from some state i to some state $j > i$. Let A_{ik} be the event that 'k ones out of i flip to zero and $i - k$ ones are not flipped' and B_{ijk} the event that '$k + j - i$ zeros out of $\ell - i$ flip to one and $\ell - j - k$ zeros are not flipped'. Note that both events are independent. The probabilities of these events are

$$\mathbf{P}\{A_{ik}\} = \binom{i}{k} p^k (1-p)^{i-k} \quad \text{and} \quad \mathbf{P}\{B_{ijk}\} = \binom{\ell - i}{k + j - i} p^{k+j-i} (1-p)^{\ell-j-k}.$$

Thus, the probability to transition from state i to j is the sum over k of the product of the probabilities of both events ($0 \le i < j \le \ell$):

$$\begin{aligned} p_{ij} &= \sum_{k=0}^{\ell} \mathbf{P}\{A_{ik}\} \cdot \mathbf{P}\{B_{ijk}\} \\ &= \sum_{k=0}^{\ell} \binom{i}{k} p^k (1-p)^{i-k} \cdot \binom{\ell - i}{k + j - i} p^{k+j-i} (1-p)^{\ell-j-k} \\ &= \sum_{k=0}^{\ell} \binom{i}{k} \binom{\ell - i}{k + j - i} p^{2k+j-i} (1-p)^{\ell+i-j-2k} \\ &= p^{j-i} (1-p)^{\ell-(j-i)} \sum_{k=0}^{\ell} \binom{i}{k} \binom{\ell - i}{k + j - i} \left(\frac{p}{1-p}\right)^{2k}. \end{aligned} \tag{B2.4.73}$$

This formula is equivalent to that of Bäck (1992, p 88). The last nonzero term of the series in (B2.4.73) is that with index $k = \min\{i, \ell - j\}$. For larger indices at least one of the binomial coefficients becomes zero. This reflects the fact that some of the events are impossible: for example, the event A_{ik} can not occur if $k > i$ because one cannot flip k ones when there are only i. Since the Markov chain stays in its current state if mutation has generated a state of worse or equal quality the probabilities of staying are

$$p_{ii} = 1 - \sum_{j=i+1}^{\ell} p_{ij}$$

for $0 \leq i < \ell$. Clearly, $p_{\ell\ell} = 1$. Since all other entries are zero the transition matrix $\mathbf{P} = (p_{ij})$ has been derived completely.

Now we are in the position to use the technique described previously. Mühlenbein (1992) used a similar method to attack this problem: in principle, he also converted the exact Markov chain to another one, that always performs less well than the original one but which is much simpler to analyze. Actually, his analysis was a pure approximation without taking into account whether the approximations yielded a lower or upper bound of the expected absorption time. However it will be shown in the following that this approach leads to an upper bound of the expectation of the first hitting time.

In the third step the original Markov chain is approximated by a simpler one that has (provable) worse performance. Recall that the key idea is to ignore all those paths that take shortcuts to state ℓ by jumping over some states in between. If the original Markov chain takes such a shortcut this move is considered deteriorating in the approximating Markov chain and it stays at its current state. Consequently, the approximating Markov chain needs more time to reach the absorbing state ℓ on average. Moreover, the approximating chain must pass all states greater than or equal to i to arrive at state ℓ when being started in state $i < \ell$.

Thus, one needs to know the transition probabilities q_{ij} of the simplified Markov chain. Actually, it is sufficient to know the values for $q_{i,i+1}$ with $i = 0, \ldots, \ell - 1$. In this case (B2.4.73) reduces to

$$q_{i,i+1} = p\,(1-p)^{\ell-1} \sum_{k=0}^{\ell} \binom{i}{k} \binom{\ell-i}{k+1} \left(\frac{p}{1-p}\right)^{2k} \tag{B2.4.74}$$

$$\geq p\,(1-p)^{\ell-1}(\ell - i). \tag{B2.4.75}$$

Expression (B2.4.74) is still too complicated. Therefore it was bounded by (B2.4.75). In principle, the approximating Markov chain was approximated again by a Markov chain with even worse performance: the probabilities to transition to the next state were decreased so that this (third) Markov chain will take an even longer time to reach state ℓ. To determine the expected time until absorption insert (B2.4.75) into (B2.4.72). This leads to

$$\mathbf{E}[\,T\,] \leq \sum_{i=0}^{\ell-1} \frac{1}{p\,(1-p)^{\ell-1}(\ell-i)} = \frac{1}{p\,(1-p)^{\ell-1}} \sum_{i=1}^{\ell} \frac{1}{i} \leq \frac{\log \ell + 1}{p\,(1-p)^{\ell-1}}. \tag{B2.4.76}$$

Evidently, the absorption time depends on the mutation probability $p \in (0, 1)$ and attains its minimum for $p^* = 1/\ell$. Then (B2.4.76) becomes (also see Mühlenbein 1992, p 19)

$$\mathbf{E}[\,T\,] \leq \ell\,(\log \ell + 1)\left(1 - \frac{1}{\ell}\right)^{1-\ell} \leq \ell\,(\log \ell + 1)\,\exp(1). \tag{B2.4.77}$$

The bound (B2.4.77) is very close to the absorption time of the original Markov chain with $p = 1/\ell$. It is clear that the optimal mutation probability for the original Markov chain will differ from $1/\ell$, but the difference is remarkably small as the numerical investigations of Bäck (1993) reveal.

B2.4.2.4 Unimodal binary functions

The notion of *unimodal* functions usually appears in probability theory (to describe the shape of probability density functions), nonlinear one-dimensional dynamics (to characterize the shapes of return maps) and in the theory of optimization of one-dimensional functions with domain $\mathbb{R}$. Since a commonly accepted definition for unimodal functions in $\mathbb{R}^\ell$ does not seem to exist, it comes as no surprise that the definition of unimodality of function with domain $\mathbb{B}^\ell$ is not unique in the literature either. Here, the following definition will be used.

Definition B2.4.2. Let f be a real-valued function with domain $\mathbb{B}^\ell$. A point $\boldsymbol{x}^* \in \mathbb{B}^\ell$ is called a *local solution* of f if

$$f(\boldsymbol{x}^*) \geq f(\boldsymbol{x}) \text{ for all } \boldsymbol{x} \in \{\boldsymbol{y} \in \mathbb{B}^\ell : \|\boldsymbol{y} - \boldsymbol{x}^*\|_1 = 1\}. \tag{B2.4.78}$$

If the inequality in (B2.4.78) is strict, then $\boldsymbol{x}^*$ is termed a *strictly local solution*. The value $f(\boldsymbol{x}^*)$ at a (strictly) local solution is called a *(strictly) local maximum* of f. A function $f : \mathbb{B}^\ell \to \mathbb{R}$ is said to be *unimodal*, if there exists exactly one local solution. □

Before determining the expected absorption time of a $(1+1)$-EA for this problem, it is useful to know whether such problems are solvable in polynomial time at all. Johnson *et al* (1988, p 86) have shown that this problem cannot be NP hard unless NP = co-NP, an event which is commonly considered very unlikely.

The *ladder problem* consists of the task of finding the maximum of the unimodal binary function

$$f(\boldsymbol{x}) = \sum_{i=1}^{\ell} \prod_{j=1}^{i} x_j$$

which is attained if all entries in vector $\boldsymbol{x}$ are set to 1. The objective function counts the number of consecutive 1s in $\boldsymbol{x}$ from left to right. Note that this function is unimodal: choose $\boldsymbol{x} \in \mathbb{B}^\ell$ such that $\boldsymbol{x} \neq \boldsymbol{x}^*$. It suffices to show that there exists a point $y \in \mathbb{B}^\ell$ with $\|\boldsymbol{x}-\boldsymbol{y}\|_1 = 1$ and $f(\boldsymbol{y}) > f(\boldsymbol{x})$. In fact, this is true: since $\boldsymbol{x} \neq \boldsymbol{x}^*$ there exists an index $k = \min\{i : x_i = 0\} \leq \ell$. Choose $\boldsymbol{y} \in \mathbb{B}^\ell$ such that $y_i = x_i$ for all $i \in \{1, \ldots, \ell\} \setminus \{k\}$ and $y_k = 1$. By construction one obtains $\|\boldsymbol{x}-\boldsymbol{y}\|_1 = 1$ and finally $f(\boldsymbol{y}) > f(\boldsymbol{x})$ since the number of consecutive 1s in $\boldsymbol{y}$ is larger than the number of consecutive 1s in $\boldsymbol{x}$. Consequently, $\boldsymbol{x}^* = (1 \ldots 1)'$ is the only point at which f attains a local maximum. Therefore f is unimodal.

To derive an upper bound on the expected number of steps to reach the global maximum consider the following decomposition of the search space: define $S_i := \{\boldsymbol{x} \in \mathbb{B}^\ell : f(\boldsymbol{x}) = i\}$ for $i = 0, 1, \ldots, \ell$. For example, for $\ell = 4$ one obtains

$$\begin{aligned} S_0 &= \{0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111\} \\ S_1 &= \{1000, 1001, 1010, 1011\} \\ S_2 &= \{1100, 1101\} \\ S_3 &= \{1110\} \\ S_4 &= \{1111\}. \end{aligned}$$

Thus, if $\boldsymbol{x} \in S_i$ then the first i bits are set correctly. Note that this grouping of states is not suited to formulate a Markov chain model with $\ell+1$ states that is equivalent to a model with 2^ℓ states. But it is possible to formulate a simplified Markov chain model with $\ell+1$ states that has worse performance than the true model. To this end assume $\boldsymbol{x} \in S_0$. Subset S_0 only can be left if the first entry mutates from zero to one. If this event occurs the Markov chain is at least in subset S_1. But it may also happen that the Markov chain transitions to any other subset S_i with $i > 1$. In the simplified model these events are not allowed: all transitions from S_0 to S_i with $i > 1$ are considered as transitions to S_1. Subset S_1 can be left only if the first entry does not mutate and the second entry flips from zero to one. In this case the Markov chain transitions at least to subset S_2. All transitions to subset S_i with $i > 2$ are considered as transitions to S_2. Analogous simplifications apply to the other subsets S_i. Since all shortcuts on the path to S_ℓ are bent back to a transition of the type S_i to S_{i+1} the expected number of trials of the simplified Markov chain is larger than the expected number of trials of the original Markov chain. The state space of the simplified Markov chain is $S = \{0, 1, \ldots, \ell\}$ where state $i \in S$ represents subset S_i. The only possible path from state 0 to state ℓ must visit all states in between in ascending order. Thus, the probability $p_{i,i+1}$ to transition from state i to $i+1$ for $i < \ell$ is the probability to flip entry $i+1$ multiplied by the (independent) probability that the first i entries remain unchanged. Thus,

$$p_{i,i+1} = p(1-p)^i$$

where $p \in (0, 1)$ denotes the probability to flip from 0 to 1 and vice versa. The expected number of steps to reach the optimum is

$$\mathbf{E}[\, T_{0,\ell} \,] = \sum_{i=0}^{\ell-1} \mathbf{E}[\, T_{i,i+1} \,] = \sum_{i=0}^{\ell-1} \frac{1}{p_{i,i+1}} = \frac{1}{p} \sum_{i=0}^{\ell-1} \left(\frac{1}{1-p} \right)^i = \frac{1-p}{p^2} [\, (1-p)^{-\ell} - 1 \,]. \qquad \text{(B2.4.79)}$$

Now insist that $p = c/\ell$ with $0 < c < \ell$. Insertion into (B2.4.79) leads to

$$\mathbf{E}[\, T_{0,\ell} \,] = \frac{\ell^2}{c^2} \left(1 - \frac{c}{\ell}\right) \left[\left(1 - \frac{c}{\ell}\right)^{-\ell} - 1 \right] \leq \ell^2 \, \frac{e^c - 1}{c^2} \qquad \text{(B2.4.80)}$$

where the rightmost expression attains its minimum for $c \approx 1.6$. In summary, it has been shown that the expected number of steps of the $(1+1)$-EA can be bounded by $O(\ell^2)$.

Let $F = \{f(\boldsymbol{x}) : \boldsymbol{x} \in \mathbb{B}^\ell\}$ be the set of function values of a unimodal function f. If the cardinality of F is bounded by a polynomial in ℓ, then it is guaranteed that the (1+1)-EA will be absorbed at the local/global solution after polynomially many trials on average, because only polynomially many improvements via one-bit mutations are possible and sufficient to reach the optimum. Such a problem was considered in the preceding example. Therefore, these problems can be excluded from further considerations. Rather, unimodal problems with $|F| = \Omega(2^\ell)$ are the interesting candidates.

By definition, each unimodal problem has at least one path to the optimum with strictly increasing function values, where consecutive points on the path differ in one bit only. Since the expected time to change a single specific bit is less than $e\ell$, an upper bound on the absorption time is the length of the path times $e\ell$. Horn *et al* (1994) succeeded in constructing paths that grow exponentially in ℓ and can be used to build unimodal problems. Consequently, the upper bound derived by the above reasoning either is too rough or indicates that polynomial bounds do not exist. It is clear that such a 'long path' must possess much structure, because the one-bit path has to be folded several times to fit into 'box' $\mathbb{B}^\ell$. One might suspect that there exist many shortcuts, by appropriate two-bit mutations, that decrease the order of the upper bound considerably. In fact, this is true. Since the analysis is quite involved only the result will be reported: the exponentially long `root2`-path is maximized after $O(\ell^3)$ function evaluations on average (see Rudolph 1996).

B2.4.2.5 Supermodular functions

Definition B2.4.3. A function $f : \mathbb{B}^\ell \to \mathbb{R}$ is said to be *supermodular* if

$$f(\boldsymbol{x} \wedge \boldsymbol{y}) + f(\boldsymbol{x} \vee \boldsymbol{y}) \geq f(\boldsymbol{x}) + f(\boldsymbol{y}) \tag{B2.4.81}$$

for all $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{B}^\ell$. If the inequality in (B2.4.81) is reversed, then f is called *submodular*. □

Evidently, if $f(\boldsymbol{x})$ is supermodular then $g(\boldsymbol{x}) := a + bf(\boldsymbol{x})$ with $a \in \mathbb{R}$ and $b \in \mathbb{R}\backslash\{0\}$ is supermodular for $b > 0$ and submodular for $b < 0$. Thus, maximization of supermodular functions is of the same difficulty as the minimization of submodular functions. For this problem class there exists a strong result.

Theorem B2.4.1 (Grötschel et al 1993, pp 310–11). Each supermodular function $f : \mathbb{B}^\ell \to \mathbb{Q}$ can be globally maximized in strongly polynomial time. □

As will be shown, it is impossible to obtain an upper bound $\widehat{T}$ on the expectation of the first hitting time that is polynomial in ℓ.

Theorem B2.4.2. There exist supermodular functions that cannot be maximized by a $(1 + 1)$-EA with a number of mutations that is upper bounded by a polynomial in ℓ.

Proof. Consider the objective function

$$f(\boldsymbol{x}) = \begin{cases} \ell + 1 & \text{if } \|\boldsymbol{x}\|_1 = \ell \\ \ell - \|\boldsymbol{x}\|_1 & \text{if } \|\boldsymbol{x}\|_1 < \ell \end{cases} \tag{B2.4.82}$$

that is easily shown to be supermodular. The state space of the $(1 + 1)$-EA can be represented by $S = \{0, 1, \ldots, \ell\}$ where each state $i \in S$ represents the number of 1s in vector $\boldsymbol{x} \in \mathbb{B}^\ell$. The absorbing state is state ℓ. It can be reached from state $i \in \{0, 1, \ldots, \ell - 1\}$ within one step with probability

$$p_{i\ell} = p^{\ell - i}\,(1 - p)^i.$$

Let the Markov chain be in some state $i \in \{0, \ldots, \ell - 1\}$. Only transitions to some state $j < i$ or to state ℓ are possible. If the Markov chain transitions to state $j < i$, then the probability to transition to state ℓ has become smaller. Thus, it would be better to stay at state i than to move to state $j < i$ although the objective function value of state j is better than the objective function value of state i. This leads to the simplified Markov chain that has better performance than the original one. Then $p_{ii} = 1 - p_{i\ell}$ and the simplified Markov chain is described completely. Thus, the expected time to transition to state ℓ from state $i < \ell$ is

$$\mathbf{E}[\,T_{i,\ell}\,] = \frac{1}{p_{i\ell}} = \frac{1}{p^{\ell - i}\,(1 - p)^i} = \ell^\ell \left(\frac{1}{\ell - 1}\right)^i \geq \ell^{\ell - i}.$$

Assuming that the initial point is drawn from a uniform distribution over $\mathbb{B}^\ell$ the average time to absorption is larger than

$$2^{-\ell}\sum_{i=0}^{\ell-1}\binom{\ell}{i}\mathbf{E}[\,T_{i,\ell}\,] \geq 2^{-\ell}\sum_{i=0}^{\ell-1}\binom{\ell}{i}\ell^{\ell-i} = \left(\frac{\ell}{2}\right)^{\ell}\sum_{i=0}^{\ell-1}\binom{\ell}{i}\left(\frac{1}{\ell}\right)^{i} = \left(\frac{\ell+1}{2}\right)^{\ell} - 2^{-\ell}.$$

Since the lower bound on the absorption time is exponential in ℓ for $\ell \geq 2$ the proof is completed. □

Of course, this result does not imply that a GA must fail to solve this problem in a polynomially bounded number of generations. It may be that some crossover operator can help. But note that the objective function (B2.4.82) is *fully deceptive* as can be easily verified owing to the sufficient conditions presented by Deb and Goldberg (1994). Fully deceptive functions are the standard examples to show (empirically) that a GA fails. B2.7.1

B2.4.2.6 Almost-positive functions

Theorem B2.4.3 (Hansen and Simeone 1986, p 270). The maximum of an almost-positive pseudo-Boolean function (i.e. the coefficients of all nonlinear terms are nonnegative) can be determined in strongly polynomial time. □

Theorem B2.4.4. There exist supermodular functions that cannot be maximized by a $(1+1)$-EA with a number of mutations that is upper bounded by a polynomial in ℓ.

Proof. Theorem B2.4.2 has shown that the objective function in equation (B2.4.82) cannot be maximized by a number of mutations that is upper bounded by a polynomial in ℓ. Note that the function in (B2.4.82) has the alternative representation

$$f(\boldsymbol{x}) = \ell - \sum_{i=1}^{\ell} x_i + (\ell+1)\prod_{i=1}^{\ell} x_i$$

revealing that this function is also almost positive. This completes the proof. □

References

Abramowitz M and Stegun I A 1984 *Pocketbook of Mathematical Functions* (Thun: Harri Deutsch)

Bäck T 1992 The interaction of mutation rate, selection, and self-adaptation within a genetic algorithm *Parallel Problem Solving from Nature, 2 (Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature, Brussels, 1992)* ed R Männer and B Manderick (Amsterdam: North-Holland) pp 85–94

——1993 Optimal mutation rates in genetic search *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 2–8

Beyer H-G 1989 *Ein Evolutionsverfahren zur mathematischen Modellierung stationärer Zustände in dynamischen Systemen* Doctoral Dissertation, HAB-Dissertation 16, Hochschule für Architektur und Bauwesen, Weimar

——1993 Toward a theory of evolution strategies: some asymptotical results from the $(1 \dagger \lambda)$-theory *Evolutionary Comput.* **1** 165–88

——1994a Towards a theory of 'evolution strategies': progress rates and quality gain for $(1 \dagger \lambda)$-strategies on (nearly) arbitrary fitness functions *Parallel Problem Solving from Nature—PPSN III (Proc. Int. Conf. on Evolutionary Computation and 3rd Conf. on Parallel Problem Solving from Nature, Jerusalem, October 1994) (Lecture Notes in Computer Science 866)* ed Yu Davidor, H-P Schwefel and R Männer (Berlin: Springer) pp 58–67

——1994b *Towards a Theory of 'Evolution Strategies': Results from the N-dependent* (μ, λ) *and the Multi-Recombinant* $(\mu/\mu, \lambda)$ *Theory* Department of Computer Science Technical Report SYS-5/94, University of Dortmund

——1995a *How GAs do* NOT *Work—Understanding GAs without Schemata and Building Blocks* Department of Computer Science Technical Report SYS-2/95, University of Dortmund

——1995b Toward a theory of evolution strategies: the (μ, λ)-theory *Evolutionary Comput.* **2** 381–407

——1995c Toward a theory of evolution strategies: on the benefit of sex—the $(\mu/\mu, \lambda)$-theory *Evolutionary Comput.* **3** 81–111

——1996a Towards a theory of evolution strategies: self-adaptation *Evolutionary Comput.* **3** 311–47

——1996b An alternative explanation for the manner in which genetic algorithms operate *Biosystems* at press

Bronstein I N and Semendjajew K A 1981 *Taschenbuch der Mathematik* (Leipzig: Teubner)
David H A 1970 *Order Statistics* (New York: Wiley)
Deb K and Goldberg D E 1994 Sufficient conditions for deceptive and easy binary functions *Ann. Math. Artificial Intell.* **10** 385–408
Fogel D B 1992 *Evolving Artificial Intelligence* PhD Thesis, University of California, San Diego
——1994 Asymptotic convergence properties of genetic algorithms and evolutionary programming: analysis and experiments *Cybernet. Syst.* **25** 389–408
——1995 *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence* (Piscataway, NJ: IEEE)
Goldberg D 1989 *Genetic Algorithms in Search, Optimization, and Machine Learning* (Reading, MA: Addison-Wesley)
Grötschel M, Lovász L and Schrijver A 1993 *Geometric Algorithms and Combinatorial Optimization* 2nd edn (Berlin: Springer)
Hansen P and Simeone B 1986 Unimodular functions *Discrete Appl. Math.* **14** 269–81
Horn J, Goldberg D E and Deb K 1994 Long path problems *Parallel Problem Solving from Nature—PPSN III (Proc. Int. Conf. on Evolutionary Computation and 3rd Conf. on Parallel Problem Solving from Nature, Jerusalem, October 1994) (Lecture Notes in Computer Science 866)* ed Yu Davidor, H-P Schwefel and R Männer (Berlin: Springer) pp 149–58
Iosifescu M 1980 *Finite Markov Processes and Their Applications* (Chichester: Wiley)
Johnson D S, Papadimitriou C H and Yannakakis M 1988 How easy is local search? *J. Comput. Syst. Sci.* **37** 79–100
Mühlenbein H 1992 How genetic algorithms really work: mutation and hillclimbing *Parallel Problem Solving from Nature, 2 (Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature, Brussels, 1992)* ed R Männer and B Manderick (Amsterdam: North-Holland) pp 15–25
Mühlenbein H and Schlierkamp-Voosen D 1993 Predictive models for the breeder genetic algorithm *Evolutionary Comput.* **1** 25–49
Rechenberg I 1994 *Evolutionsstrategie '94* (Stuttgart: Frommann-Holzboog)
Rudolph G 1996 How mutation and selection solve long path problems in polynomial expected time *Evolutionary Comput.* **4** at press
Schwefel H-P 1995 *Evolution and Optimum Seeking* (New York: Wiley)

Further reading

1. Arnold B C, Balakrishnan N and Nagaraja H N 1992 *A First Course in Order Statistics* (New York: Wiley)

   As does the book of David (1970), this course gives a good introduction into order statistics, which builds the mathematical basis for truncation selection.

2. Beyer H-G 1992 *Towards a Theory of 'Evolution Strategies'. Some Asymptotical Results from the* $(1 \dot{+} \lambda)$*-Theory* Department of Computer Science Technical Report SYS-5/92, University of Dortmund

   In this report the derivations for the $(1 \dot{+} \lambda)$ theory on noisy fitness data can be found.

3. Beyer H-G 1994 *Towards a Theory of 'Evolution Strategies': Results from the* N*-dependent* (μ, λ) *and the Multi-Recombinant* $(\mu/\mu, \lambda)$ *Theory* Department of Computer Science Technical Report SYS-5/94, University of Dortmund

   This report contains the 'hairy details' of the progress rate theory for (μ, λ) and $(\mu/\mu, \lambda)$ ESs as well as the derivations for the differential geometry approach.

4. Beyer H-G 1995 *Towards a Theory of 'Evolution Strategies': the* $(1, \lambda)$*-Self-Adaptation* Department of Computer Science Technical Report SYS-1/95, University of Dortmund

   This report is devoted to the theory of $(1, \lambda)$ σ selfadaptation and contains the derivations of the results presented in the article by Beyer (1996).

5. Michod R E and Levin B R (eds) 1988 *The Evolution of Sex: an Examination of Current Ideas* (Sunderland, MA: Sinauer)

   Concerning the current ideas on the benefits of recombination in biology, this book reflects the different hypotheses on the evolution of sex. Biological arguments and theories should receive more attention within the EA theory.

B2.5 Schema processing

Nicholas J Radcliffe

Abstract

From the earliest days, genetic algorithms have been analyzed in terms of their effects on *schemata*—groups of individuals with shared values for some genes. This section presents the basic definitions and results from schema analysis together with a critical discussion. Topics covered include genes, alleles, schemata, the schema theorem, building blocks, nonlinearities, cardinality, linkage, and generalizations of the basic schema-processing framework. Particular emphasis is given to careful interpretation of the results, including the much-debated issue of so-called *implicit parallelism*.

B2.5.1 Motivation

Schema analysis was invented by John Holland, and presented to the world in his book of 1975, as a possible basis for a theory of genetic algorithms. One of Holland's basic motivations and beliefs was that complex problems are most easily solved by breaking them down into a set of simpler, more tractable subproblems, and this belief is visible both in Holland's writing on genetic algorithms and in his conception of schema analysis.

Loosely speaking, schema analysis depends on describing a solution to a search problem as a set of assignments of values (*alleles*) to variables (*genes*). A schema can then be viewed as a partial solution, in which only some of the variables have specified values. Various measures of the quality of such a partial solution are used, mostly based on sampling the different solutions obtained by completing the schema with different variable assignments. Schemata† thus provide a way of decomposing a complex search problem into a hierarchy of progressively simpler ones in which the simplest level consists of single variable assignments. Informally, the idea is that a genetic algorithm tackles the simpler problems first, and that these partial solutions are then combined (especially through *recombination*, also known as crossover) into more complex solutions until eventually the complete problem is solved.

Schema analysis is primarily concerned with the dynamics of schemata over the course of the run of a genetic algorithm, and its most famous (if limited) result, the schema theorem, provides a partial description of those dynamics.

B2.5.2 Classical schema analysis

Having described the motivation for the idea of schemata, we now derive the schema theorem. This requires some careful definitions, which now follow.

B2.5.2.1 Basic definitions

Definition B2.5.1 (representation, chromosome, gene, and allele). Let $\mathcal{S}$ be a *search space*, i.e. a collection of objects over which search is to be conducted. Let $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n$ be arbitrary finite sets, and let B2.1.2

$$I = \mathcal{A}_1 \times \mathcal{A}_2 \times \cdots \times \mathcal{A}_n.$$

† 'Schemata' has traditionally been the preferred plural form of schema, though 'schemas' is fast gaining favor within the research community.

Finally let

$$g : I \longrightarrow \mathcal{S}$$

be a function mapping vectors in I to solutions in the search space. Then I and g are together said to form a *representation* of $\mathcal{S}$. I is called a *representation space* and g is known as a *growth function.*

The members of I are called *chromosomes* or *individuals* (and less commonly *genomes* or *genotypes*). The sets $\mathcal{A}_i$ from which I is composed are called *allele sets* and their members are called *alleles.* A chromosome $\boldsymbol{x} \in I$ can, of course, be expanded to be written as

$$(x_1, x_2, \ldots, x_n) \in \mathcal{A}_1 \times \mathcal{A}_2 \times \cdots \times \mathcal{A}_n$$

or as the string $x_1 x_2 \ldots x_n$. The components, x_i, of $\boldsymbol{x}$, when treated as variables, are known as *genes*, so that the ith gene takes values from the allele set $\mathcal{A}_i$. The position, i, of a gene on the chromosome is known as its *locus.* □

Example B2.5.1 (representation). Let $\mathcal{S} = \mathbb{N}_{10} = \{0, 1, \ldots, 9\}$ be a search space. This can be represented with the two-dimensional representation space $I = \mathbb{N}_2 \times \mathbb{N}_5$, so that chromosomes are ordered pairs of two integers, the first of which is binary and the second of which is in the range 0–4. A possible growth function for this case would be

$$g(a, b) = 5a + b.$$

The first gene, which has locus 1, is binary and has alleles 0 and 1. The second gene, which has locus 2, has cardinality 5, and has alleles 0, 1, 2, 3, and 4. □

In many representations, all the genes have the same cardinality and a common allele set. Historically, binary representations have been particularly popular, in which all genes have only the alleles 0 and 1.

Definition B2.5.2 (schema). Let $I = \mathcal{A}_1 \times \mathcal{A}_2 \times \cdots \times \mathcal{A}_n$ be a representation space. For each allele set $\mathcal{A}_i$, define the extended allele set $\mathcal{A}_i^\star$ by

$$\mathcal{A}_i^\star = \mathcal{A}_i \cup \{\star\}$$

where $\star$ is known as the 'don't care' symbol. Then a *schema* is any member of the set Ξ defined by

$$\Xi = \mathcal{A}_1^\star \times \mathcal{A}_2^\star \times \cdots \times \mathcal{A}_n^\star$$

i.e. a chromosome in which any subset of the alleles may be replaced with the 'don't care' symbol $\star$. A schema $\xi = (\xi_1, \xi_2, \ldots, \xi_n)$ describes a set of chromosomes which have the same alleles as ξ at all the positions i at which $\xi_i \neq \star$, i.e.

$$\xi = \{x \in I | \forall i \in \{1, 2, \ldots, n\} : (\xi_i = x_i \text{ or } \xi_i = \star)\}.$$

The positions at which ξ is not $\star$ are called its *defining positions.* The *order*, $o(\xi)$, of a schema ξ is the number of defining positions it contains, and its *defining length*, $\delta(\xi)$, is the distance between its first and last defining positions. □

Schemata are also known variously as *hyperplanes* (using the geometric analogy) and *similarity templates.* The members of a schema are generally referred to as *instances.*

Example B2.5.2 (schema). Let $I = \mathbb{N}_3^5$ be the set of vectors of five ternary digits. Then the schema $2\star10\star$ is given by

$$2\star10\star = \{20100, 20101, 20102, 21100, 21101, 21102, 22100, 22101, 22102\}$$

and has order three and defining length three. □

Definition B2.5.3 (fitness function). Let $\mathcal{S}$ be a search space, $F : \mathcal{S} \longrightarrow \mathbb{R}_+$ be an objective function, and $g : I \longrightarrow \mathcal{S}$ be a growth function for the representation space I. Then any function

$$f : I \longrightarrow \mathbb{R}_+$$

with the property that

$$f(x) = \max_I f \iff F(g(x)) = \operatorname*{opt}_{g(I)} F$$

where opt is min if F is to be minimized, and max if F is to be maximized, will be called a *fitness function.* □

Example B2.5.3 (fitness function). Let $F : [-1, 1] \times [-1, 1] \longrightarrow [-2, 0]$ be an objective function for minimimization, defined by

$$F(x, y) = x^2 + y^2 - 2$$

with optimum $F(0, 0) = -2$. Let the representation space be $\mathbb{N}_{11} \times \mathbb{N}_{11}$, with growth function

$$g(a, b) = \left(\frac{a}{5} - 1, \frac{b}{5} - 1\right).$$

Then a possible fitness function is given by

$$f(a, b) = 3 - F(g(a, b)).$$

Notice that, as is commonly the case, f here is a monotonic transformation of $F \circ g$. □

B2.5.2.2 The schema theorem

Theorem B2.5.1 (the schema theorem). Let ξ be any schema over a representation space I being searched by a traditional *genetic algorithm* using *fitness-proportional selection*, specified recombination and mutation operators, and *generational update*. Let $N_\xi(t)$ denote the number of instances of the schema ξ present in the population at generation t. Then B1.2, C2.2 C2.7

$$\langle N_\xi(t+1) \mid N_\xi(t) \rangle \geq N_\xi(t) \frac{\hat{f}_\xi(t)}{\bar{f}\,(t)} [1 - D_c(\xi)][1 - D_m(\xi)]$$

where:

- $\langle A|B \rangle$ denotes the conditional expectation value of A given B
- $\hat{f}_\xi(t)$ is the *observed fitness* of the schema ξ at generation t, defined by

$$\hat{f}_\xi(t) = \frac{1}{N_\xi(t)} \sum_{x \in \xi \cap \mathcal{B}(t)} f(x)$$

  where individuals occurring more than once in the population $\mathcal{B}(t)$ contribute more than once to the average; that is, $\hat{f}_\xi(t)$ is the mean fitness of all chromosomes in the population that are members of ξ
- $\bar{f}\,(t)$ is the mean fitness of the entire population at generation t
- $D_c(\xi)$ and $D_m(\xi)$ are upper bounds on the disruptive effect on schema membership of the chosen crossover and mutation operators respectively (see below).

Proof. Let $\mathcal{B}(t)$ be the population at generation t, remembering that this is a *bag*, rather than a set (i.e. repetition is allowed), and let $\mathcal{B}_\xi(t)$ denote the (bag of) members of the population that are instances of ξ. Using any recombination operator that produces two children from two parents, the total number of parents contributing to a generation is clearly the same as the number of children, i.e. the (fixed) population size. Under proportional selection, the expected number of times an individual x will act as a parent for the next generation $t + 1$ is $f(x)/\bar{f}\,(t)$. Therefore, the expected number of times individuals from $\mathcal{B}_\xi(t)$ will act as parents is $\sum_{x \in \mathcal{B}_\xi(t)} f(x)/\bar{f}\,(t) = N_\xi(t)\hat{f}_\xi(t)/\bar{f}\,(t)$. A child having a parent $x \in \xi$ will be guaranteed to be a member of that schema also provided that neither recombination nor mutation acts in such a way as to destroy that schema membership. Therefore, since $D_c(\xi)$ is, by definition, an upper bound on the probability that crossover will be applied and will cause a parent in ξ to produce a child not in ξ, and $D_m(\xi)$ is the corresponding bound for mutation, it is clear that

$$\langle N_\xi(t+1) \mid N_\xi(t) \rangle \geq N_\xi(t) \frac{\hat{f}_\xi(t)}{\bar{f}\,(t)} [1 - D_c(\xi)][1 - D_m(\xi)]$$

which is the required result. □

Corollary B2.5.1 (the schema theorem for a simple genetic algorithm). Consider a traditional genetic algorithm as above, using chromosomes with n genes, in which the chosen recombination operator is *one-point crossover*, applied with probability p_c, and the chosen mutation operator is *point mutation*, in which each gene's value is altered with independent probability p_m. Then $D_c(\xi) = p_c\delta(\xi)/(n-1)$ and $D_m(\xi) = 1 - (1-p_m)^{o(\xi)}$ act as upper bounds for the disruptive effect of recombination and mutation respectively. For small p_m, the latter bound is well approximated by $D_m(\xi) = p_m o(\xi)$, and using this approximation the schema theorem for a traditional genetic algorithm is C3.3.1, C3.2.1

$$\left\langle N_\xi(t+1) \mid N_\xi(t) \right\rangle \geq N_\xi(t)\frac{\hat{f}_\xi(t)}{\bar{f}\,(t)}\left[1 - p_c\frac{\delta(\xi)}{n-1}\right]\left[1 - p_m o(\xi)\right].$$

Proof. One-point crossover can only disrupt schema membership if the cross point falls within the defining region of the schema (i.e. between the first and last defining positions). Assuming that the cross point is chosen uniformly, the probability of this is simply the proportion of possible cross points that lie within the defining region, which is $\delta(\xi)/(n-1)$. This establishes that an upper bound on the disruptiveness of one-point crossover is given by $D_c(\xi) = p_c\delta(\xi)/(n-1)$, as required.

Point mutation can only disrupt schema membership if it alters at least one defining position. The probability of none of the defining positions of a schema being affected by point mutation is, by definition, $(1-p_m)^{o(\xi)}$, so the disruption coefficient is $D_m(\xi) = 1 - (1-p_m)^{o(\xi)}$. For $p_m \ll 1$, this is well approximated by $p_m o(\xi)$ as required. □

We will shortly consider the significance of the schema theorem, and some of its interpretations. First, however, it is worth making a few notes on its status, scope, and form.

Status. The value and significance of the schema theorem is keenly debated. Extreme positions range from sceptics who view the schema theorem as having little or no value (Mühlenbein 1992) to those who view it as 'the fundamental theorem of genetic algorithms' (Goldberg 1989c). In fact, as the above proof shows, the schema theorem is a simple, little result that can be proved, in various forms, for a variety of 'genetic algorithms'.

Selection. In the form shown, the schema theorem only applies to evolutionary algorithms using fitness-proportionate selection, which is described by the term $\hat{f}_\xi(t)/\bar{f}\,(t)$. However, it is easy to substitute terms describing most of the other selection methods commonly used in genetic algorithms (see e.g. Goldberg and Deb 1990, Hancock 1994). It is, however, *not* straightforward to extend the theorem to selection methods that depend on the fitness of the offspring produced. A particular consequence of this is that the $(\mu+\lambda)$ and (μ,λ) selection methods typically used in *evolution strategies* (Bäck and Schwefel 1993) cannot easily be incorporated in the framework of the schema theorem. The observation that substituting such selection mechanisms appears in practice to alter the behavior of a genetic algorithm relatively little might be taken as evidence that the schema theorem captures relatively little of the behavior of genetic algorithms. C2.4.4, B1.3

Other move operators. The form in which the schema theorem is presented above makes clear that it is applicable to any move operators, provided that suitable bounds can be derived for their disruptiveness. Other operators for which bounds have been derived include uniform crossover (Syswerda 1989, Spears and De Jong 1991) and partially matched crossover (PMX; Goldberg and Lingle 1985).

More precise forms of the theorem. The theorem can be tightened in a number of ways, two of which are noted here. First, many recombination operators have the property of *respect* (Radcliffe 1991). A recombination operator is respectful if whenever both parents are members of a schema this is also the case for both of the children it produces. This observation allows a tighter bound for $D_c(\xi)$ to be obtained if the probability of mating between two members of the same schema can be quantified. For one-point crossover, and the simple but noisy *roulette-wheel* implementation of fitness-proportionate selection, this more precise bound is C2.2

$$D_c(\xi) = p_c\left[1 - \frac{\hat{f}_\xi(t)}{\bar{f}\,(t)}\right]\frac{\delta(\xi)}{n-1}$$

which is essentially the value used by Holland (1975).

Secondly, if precise values are used instead of bounds for the disruption coefficients $D_c(\xi)$ and $D_m(\xi)$, and terms are added to describe the possibility of new creating new instances of schemata from parents that are not instances, the schema theorem can be turned from an inequality to an equality. The first attempt at this was that by Bridges and Goldberg (1987). Later, Nix and Vose (1991) took this further by writing down the exact function describing the expected transformation from one generation to the next. This formulation has become known as the *Vose model*. C3.3.1.3

Linkage and disruption. Consider two schemata of order two, one which is defined over adjacent positions, and the other of which has defining positions at opposite ends of the chromosome. It is clear that the first schema, having shorter defining length, is much less likely to be disrupted by one-point crossover than is the second. The degree of compactness of a schema, relative to its order, is referred to as its *linkage*, shorter schemata being tightly linked, and longer ones being loosely linked.

One of the reasons for making a distinction between the identity (or meaning) of a gene and its locus (position) on the chromosome is that, in Holland's original conception of genetic algorithms, the locus of a gene was intended to be itself subject to adaptation. The idea here is that a chromosome $(4, 2, 3, 6)$ is replaced by the locus-independent description $((1, 4), (2, 2), (3, 3), (4, 6))$, where the first number in each pair indicates the gene, and the second its value (the allele). Clearly, under this description, the positions of the genes on the chromosome may be permuted without altering the solution represented by the individual. Applying the same idea to schemata, a long, loosely linked schema such as $((1, 4), (2, \star), (3, \star), (4, 6))$ is equivalent to the tightly linked schema $((1, 4), (4, 6), (2, \star), (3, \star))$.

Holland's intention was that such locus-independent representations be used, and that a third operator, *inversion*, be introduced to alter the linkage of chromosomes by randomly reversing segments. (Notice again that this does *not* change the solution described by the chromosome.) Inversion would be applied relatively infrequently, to 'mutate' the linkage of chromosomes. Under this scheme, when two parents are brought together for recombination, one is temporarily reordered to match the linkage of the other, so that, denoting crossover by $\otimes$, and marking the cross point with $|$,

$$\begin{array}{c} \\ \otimes \end{array} \begin{array}{l} ((4,6),(1,4), \\ ((2,3),(1,5), \end{array} \begin{array}{|l} (3,3),(2,2)) \\ (4,1),(3,3)) \end{array} \longmapsto \begin{array}{c} \\ \otimes \end{array} \begin{array}{l} ((4,6),(1,4), \\ ((4,1),(1,5), \end{array} \begin{array}{|l} (3,3),(2,2)) \\ (3,3),(2,3)) \end{array}.$$

The initial result of this recombination is shown below, followed by the final result, in which the linkage of the second child is restored to that of the second parent:

$$\ldots = \begin{array}{l} ((4,6),(1,4), \\ ((4,1),(1,5), \end{array} \begin{array}{|l} (3,3),(2,3)) \\ (3,3),(2,2)) \end{array} = \begin{array}{l} ((4,6),(1,4), \\ ((2,2),(1,5), \end{array} \begin{array}{|l} (3,3),(2,3)) \\ (4,1),(3,3)) \end{array}.$$

This subtle idea suggested a mechanism whereby the linkage of solutions could itself be subject to adaptation. Although there is no direct fitness benefit gained by moving together genes with coadapted values, there is an indirect benefit in that these coadapted values are more likely to be passed on to children together. The hope was that, over a number of generations, the linkage of genes across the population would come to reflect the gene interdependencies, allowing more efficient search.

Elegant and persuasive though the idea of selfadaptive linkage is, neither early nor later work (Cavicchio 1970, Radcliffe 1990) has managed to demonstrate a clear benefit from using inversion and locus-independent representations. This is widely interpreted as indicating that current practice uses runs too short (covering too few generations) for a second-order selective pressure, such as is available from good linkage, to offer a significant performance benefit (Harik and Goldberg 1996). The belief of many workers that, as problem complexities increase, inversion will experience a renaissance is evidenced by continuing discussions of and research into relinking mechanisms (Holland 1992, Weinholt 1993, Harik and Goldberg 1996).

B2.5.3 Interpretations of the schema theorem

As noted above, the schema theorem is a very simple, little theorem, but one that is the source of much debate. Most of that debate centers not on the theorem itself, but on its interpretations (and misinterpretations). It is to these interpretations that we now turn our attention.

B2.5.3.1 Building blocks

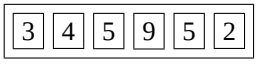
One of the fundamental beliefs that underpins much interpretation of the schema theorem is that genetic algorithms process not only individual chromosomes, but also—implicitly—schemata. Indeed, the schema theorem is pointed to as evidence of that processing, because it (potentially) gives a partial description of the dynamics of each schema instantiated in the population. (It formally gives a description of all schemata, even those not instantiated in the population, but its prediction for these is trivial.) We shall shortly consider a quantitative argument pertaining to schema processing, bringing in the concept of *implicit parallelism.* First, however, we will examine the notion of a *building block.*

As noted earlier, one of Holland's motivations for genetic algorithms and the formalism of schemata is the desire to solve complex problems by combining the solutions to simpler subproblems. Suppose that we have a problem for which a target solution is represented as $(0, 1, 1, 7, 8, 2, 3, 4)$, using a denary representation, which we will abbreviate to 01178234, following common practice. Suppose further that this problem contains two subproblems, the solutions to which are represented by nonoverlapping sets of genes. For example, suppose that the first subproblem is represented by the second and third genes, so that its solution can be described by the schema $\star 11 \star\star\star\star\star$, and that the second uses genes 5, 7, and 8, and is solved by members of the schema $\star\star\star\star 8 \star 34$. Clearly the solutions to these two subproblems are compatible (or *noncompeting*) in the sense that a single chromosome can combine both, by being a member of the higher-order schema $\star 11 \star 8 \star 34$. One of the principal functions of crossover is widely perceived to be to effect exactly such bringing together of partial solutions, by recombining one parent which is an instance of the first schema, and a second which instantiates the second, to produce a child that is a member of both schemata. At the simplest level, a *building block* can be thought of as a solution to a subproblem of the type described above that can be expressed as a schema, particularly if that schema has short defining length.

The idea of building blocks goes to the heart of the motivations for schema analysis, and is worth careful consideration. The key to being able to exploit a decomposition of a complex problem into simpler subproblems is that the solutions of those subproblems are to some extent independent. To illustrate this, consider a familiar attaché case, with six ten-digit dials arranged in two blocks of three:

This arrangement allows the complete problem of finding the correct six digits to open the case to be decomposed into two subproblems, one of opening the left-hand lock and another of opening the one on the right. Each subproblem has $10^3 = 1000$ possible solutions, so that, even using exhaustive search, a maximum of 2000 combinations needs to be tried to open both locks: the solutions to the two subproblems are independent.

Consider now an alternative arrangement, in which all six digits control a single lock (or equivalently both locks):



Although the problem can still formally be decomposed into two subproblems, that of finding the first three digits, and that of finding the second, the decomposition is no longer helpful, because finding the 'correct' solution to the first problem is in this case entirely dependent on having the correct solution to the second problem, so that now fully $10^6 = 1\,000\,000$ solutions must be considered. (Attaché case manufacturers take note!) Conversely, if all six dials are attached to separate locks, only 60 trials are needed to guarantee opening the case.

In mathematical terms, the decomposability of a problem into subproblems is referred to as *linear separability,* and models satisfying this condition are known as *additive.* In biology, the term *epistasis* is used to describe a range of nonadditive phenomena (Holland 1975), and this terminology is often used to describe nonlinearities in fitness functions tackled with genetic algorithms (Davidor 1990).

In the context of genetic algorithms, the potential for crossover to succeed in assembling chromosomes representing good solutions through recombining solutions containing useful building blocks is thus critically dependent on the degree of linear separability with respect to groups of genes present in the chosen representation. A key requirement in constructing a useful representation is thus to have regard to the degree to which such separability can be achieved. It should, however, be noted that complete separability is certainly *not* required, and there is abundant evidence that genetic algorithms are able to

cope with significant degrees of epistasis, from both studies of artificial functions (e.g. Goldberg 1989a, b) and those of complex, real-world problems (e.g. Hillis 1990, Davis 1991). This is fortunate, as only the simplest problems exhibit complete linear separability, and in such cases a genetic algorithm is far from the most efficient approach (see e.g. Schwefel 1995).

The description of building blocks given by Goldberg (1989c), which was for a long period the only introduction to genetic algorithms accessible to many people, is slightly different. He defines building blocks as 'short, low-order, and highly fit schemata' (p 41). Goldberg refers to the *building block hypothesis,* a term widely used but rarely defined. Goldberg's own statement of the building block hypothesis is:

> Short, low-order, and highly fit schemata are sampled, recombined, and resampled to form strings of potentially higher fitness.

He goes on to suggest that:

> Just as a child creates magnificent fortresses through the arrangement of simple blocks of wood, so does a genetic algorithm seek near optimal performance through the juxtaposition of short, low-order, high-performance schemata, or building blocks.

While these descriptions accurately reflect the intuitive picture many have of how genetic algorithms tackle complex problems, it is important to note that the building block hypothesis is exactly that—a hypothesis. Not only has it not been proved: it is not even precise enough in the form quoted to admit the possibility of proof. While it is not difficult to produce related, sharper hypotheses that are falsifiable, there are reasons to doubt that any strong form of the building block hypothesis will ever be proved. Some of this doubt arises from studies of simple, linearly separable functions, such as one-max (or 'counting ones') and Holland's *royal roads* (Mitchell *et al* 1992), which, though superficially well suited to solution by genetic algorithms, have proved more resistant than many predicted (Forrest and Mitchell 1992). B2.7.5

In considering the building block hypothesis as stated above, it is also important to be careful about what is meant by the fitness of a schema. This relates directly to the degree of linear separability of the problem at hand with respect to the chosen representation. Recall that the measure used in the schema theorem is the *observed* fitness $\hat{f}_\xi(t)$, i.e. the average fitness of those individuals in the current population that are members of ξ. Except in the special case of a random population, this measure is potentially very different from what is sometimes called the *static* or *true* fitness of a schema, which we will denote $\bar{f}_s$, defined as the average over all of its possible instances. This is quite widely perceived as a problem, as if the ideal situation would pertain if the observed fitness, $\hat{f}_\xi(t)$, formed an unbiased estimator for $f_s(\xi)$ (which is in general only the case during the first generation, assuming this is uniformly generated). In fact, except for the case of truly linearly separable (additive) functions, it is far from clear that this would be desirable. It seems more likely that it is the very ability of genetic algorithms to form estimates biased by the values of other genes found in solutions of above-average performance in the population that allows very good solutions often to be found in cases where the objective function is significantly nonlinear.

B2.5.3.2 Implicit parallelism and schema dynamics

Perhaps the most controversial aspect of schema analysis is the notion of *implicit parallelism,* and the arguments about representation cardinality that derive from it. We now carefully examine this idea.

Consider a representation of dimension n, i.e. with n genes. It is easy to see that, regardless of the cardinality of its genes, every chromosome $x \in I$ is a member of 2^n schemata. This is because any subset of its alleles can be replaced with a $\star$ symbol to yield a schema to which x belongs, and there are 2^n such subsets. (We shall refer to this as the *degree* of implicit parallelism.) Depending on the similarity between different members of the population, this means that a population of size λ contains instances of between 2^n and $\lambda 2^n$ different schemata. This leads to the idea that genetic algorithms *process* schemata as well as individual strings—a phenomenon Holland called *implicit parallelism*†. The schema theorem, in any of its forms, is sometimes produced as evidence for this proposition, since it gives potentially nontrivial predictions for every schema instantiated in the population.

† The original term, intrinsic parallelism, is now disfavored because it may be confused with the amenability of genetic algorithms to implementation on parallel computers.

The notion of implicit parallelism has been used to argue that representations based on genes with few alleles are superior to those with genes of higher cardinality, through what Goldberg (1989c) has termed the *principle of minimal alphabets.* The argument is that since each chromosome is a member of 2^n different schemata, and accepting that genetic algorithms 'process' such schemata, the number of schemata processed is maximized by representations that maximize 2^n. Since the number of genes required for any given size of representation is inversely related to the number of alleles per gene, this argues for large numbers of low-cardinality genes—ideally all-binary genes.

A number of critiques of this argument have been put forward.

(i) Antonisse (1989) pointed out that Holland's definition of schemata was not the only one possible. Suppose that rather than extending allele sets by a single 'don't care' symbol to obtain schemata, a 'schema' is allowed to specify any subset of the available alleles for any position. An example of such an extended schema for base-5 strings might be

$$1\left\{\begin{array}{c}1\\2\\4\end{array}\right\}11 = \{1111, 1211, 1411\}.$$

It is easy to see that the proof of the schema theorem is unaffected by this broadening of the definition of schemata. However, the degree of implicit parallelism computed with respect to the new schemata is now much higher for nonbinary representations—indeed higher than for binary representations.

(ii) Vose and Liepins (1991b) and Radcliffe (1991) independently went further, and both showed that, provided the schema theorem is written in terms of general disruption coefficients as above, arbitrary subsets of the search space (called *formae* by Radcliffe, and *predicates* by Vose) also obey the schema theorem. If the notion of implicit parallelism is taken seriously, this suggests that the degree is independent of the representation, and always equal to $2^{|I|-1}$, because there each individual is a member of $2^{|I|-1}$ subsets of I. This is plainly not a helpful notion. While it may be argued that not all of these subsets are 'usefully processed', because their level of disruption is high, many plainly have extremely low levels of disruption.

(iii) Perhaps the most fundamental reason for doubting the claims of inherent superiority of low-cardinality representations comes from arguments now referred to as the *no-free-lunch theorem* (Wolpert and Macready 1995, Radcliffe and Surry 1995). Broadly speaking, these show that only by exploiting some knowledge of the function being tackled can any search algorithm have any opportunity to exceed the performance of enumeration. The number of possible binary representations of any search space is combinatorially larger than the number of points in that space, specifically $\geq N!$, where $N = |\mathcal{S}|$, assuming that all points in $\mathcal{S}$ are to be represented. It is easy to show that most of these representations will result in genetic algorithms that perform worse than enumeration on any reasonable performance measure, so the problem of choosing a representation is clearly much more strongly dependent on selecting meaningful genes (which Goldberg (1989c), calls the *principle of meaningful building blocks*) than on the choice of representation cardinality.

Although much of the theoretical work concerning genetic algorithms continues to concentrate on binary representations, applications work is increasingly moving away from them. Readers interested to learn more about implicit parallelism are referred to the book by Holland (1975), where it is discussed in more detail, and the work of Goldberg (1989c) and Bertoni and Dorigo (1993), who update and expand upon Holland's famous N^3 estimate of the number of schemata implicitly processed by a genetic algorithm.

B2.5.4 The k-armed bandit analogy and proportional selection

A question that much concerned Holland in devising genetic algorithms was the *optimal allocation of trials*. This is quite natural, because the fundamental question any adaptive search algorithm has to address is: on the basis of the information collected so far from the points sampled from the search space $\mathcal{S}$, which point should be sampled next?

Holland tackled this problem by considering an analogy with gambling on machines related to *one-armed bandits*. Suppose a bandit machine has two arms (a *two-armed bandit*), and that they have different average payoffs (rates of return), but that it is not known which arm is which. How should *trials* be allocated between the arms to minimize expected cumulative losses? This is a well-posed decision problem

that can be solved with standard probabilistic methods. Clearly both arms need to be sampled initially, and after a while one arm will typically be seen to be showing a higher observed payoff than the other. The subtlety in solving the problem arises in trading off the expected benefit of using the arm with higher observed payoff against the danger that statistical error is responsible for the difference in performance, and in fact the arm with the lower performance has the higher true average payoff. Roughly speaking, Holland showed that minimum losses are almost realized by a strategy of biasing further trials in favor of the arm with better observed payoff as an exponentially increasing function of the observed performance difference. Extending this analysis to a bandit with k arms, and then replacing arms with schemata, Holland argued that the allocation of trials to schemata should be proportionate to their observed performance, and this was the original motivation for fitness-proportionate selection mechanisms. However, a few points should be noted in this connection.

- In most problems tackled with genetic algorithms, the concern is not with maximizing *cumulative* performance,† but rather with maximizing either the rate of improvement of solutions or the quality of the best solution that can be obtained for some fixed amount of computational resource: Holland's analysis of optimal allocation of trials is not directly relevant to this case.
- In most cases, people are relatively unconcerned with monotonic transformations of the objective function, since these affect neither the location of optima nor the ranking of solutions. This freedom to apply monotonic transformations is often used in producing a fitness function over the representation space from the objective function over the search space. Relating selection pressure directly to the numeric fitness value therefore seems rather arbitrary.
- Holland's analysis treats the payoffs from different schemata as independent, but in fact they are not.

In practice, the most widely used selection methods today either discard actual fitness values completely, and rely only on the relative rank of solutions to determine their *sampling rates*, or scale fitness values on a time-varying basis designed to maintain selection pressure even when *fitness ratios* between different solutions in the population become small. C2.4 C2.2

B2.5.5 Extensions of schema analysis

Schema analysis has been extended in various ways. As noted above, Nix and Vose (1991) developed an extension based on writing down the exact transition matrix describing the expected next generation. For more details of this extension, the interested reader is referred to Section C3.3.1.3, and the articles by Vose and Liepins (1991a) and Vose (1992). C3.3.1.3

Other generalizations start from the observation that schema analysis depends largely on the assumptions that all vectors in the representation space I correspond to valid solutions in $\mathcal{S}$, and that the move operators are defined by exchanging gene values between chromosomes (crossover) and altering small numbers of gene values (mutation). Many of the 'genetic algorithms' in common use satisfy neither of these assumptions.

Representations in which all allocations of alleles to genes represent valid solutions are said to be *orthogonal*. Familiar examples of nonorthogonal representations include permutation-based representations (such as most of those for the *traveling salesman problem*), and most representations for constrained optimization problems. Recombination operators not based simply on exchanging gene values (using a 'crossover mask') include partially matched crossover (which Goldberg and Lingle (1985) have analyzed with o-schemata, a form of generalized schemata), blend crossover (Eshelman and Schaffer 1992), random respectful recombination (Radcliffe 1991), and line crossover (Michalewicz 1992), to name but a few. Non-point-based mutation operators include Gaussian ('creep') mutation (Bäck and Schwefel 1993) and binomial minimal mutation (Radcliffe and Surry 1994), as well as the various hillclimbing operators, which are sometimes regarded as generalizations of mutation operators. G9.5

Schema analysis does not naturally address any of these cases. A more general formalism, known as forma analysis (see e.g. Radcliffe 1991, Radcliffe and Surry 1994) uses characterizations of move operators and representations to provide a more general framework, allowing insights to be transferred between problems and representations of different types.

† It is interesting to note that the *on-line* and *off-line* performance measures (De Jong 1975) used in much early work on genetic algorithms *do* relate to cumulative performance.

References

Antonisse J 1989 A new interpretation of schema notation that overturns the binary coding constraint *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, WA, 1989)* ed J D Schaffer (San Mateo,CA: Morgan Kaufmann)

Bäck T and Schwefel H-P 1993 An overview of evolutionary algorithms for parameter optimisation *Evolutionary Comput.* **1** 1–24

Bertoni A and Dorigo M 1993 Implicit parallelism in genetic algorithms *Artificial Intell.* **61** 307–14

Bridges C and Goldberg D E 1987 An analysis of reproduction and crossover in a binary-coded genetic algorithm *Proc. 2nd Int. Conf. on Genetic Algorithms (Pittsburgh, PA, 1987)* ed J J Grefenstette (Hillsdale, NJ: Erlbaum)

Cavicchio D J 1970 *Adaptive Search Using Simulated Evolution* PhD Thesis, University of Michigan

Davidor Y 1990 Epistasis variance: suitability of a representation to genetic algorithms *Complex Syst.* **4** 369–83

Davis L 1991 *Handbook of Genetic Algorithms* (New York: Van Nostrand Reinhold)

De Jong K A 1975 *An Analysis of the Behaviour of a Class of Genetic Adaptive Systems* PhD Thesis, University of Michigan

Eshelman L J and Schaffer D J 1992 Real-coded genetic algorithms and interval schemata *Foundations of Genetic Algorithms 2* ed D Whitley (San Mateo, CA: Morgan Kaufmann)

Forrest S and Mitchell M 1992 Relative building block fitness and the building block hypothesis *Foundations of Genetic Algorithms 2* ed D Whitley (San Mateo, CA: Morgan Kaufmann)

Goldberg D E 1989a Genetic algorithms and Walsh functions: Part I, a gentle introduction *Complex Syst.* **3** 129–52

——1989b Genetic algorithms and Walsh functions: Part II, deception and its analysis *Complex Syst.* **3** 153–71

——1989c *Genetic Algorithms in Search, Optimization and Machine Learning* (Reading, MA: Addison-Wesley)

Goldberg D E and Deb K 1990 A comparative analysis of selection schemes used in genetic algorithms *Foundations of Genetic Algorithms* ed G J E Rawlins

Goldberg D E and Lingle R Jr 1985 Alleles, loci and the traveling salesman problem *Proc. Int. Conf. on Genetic Algorithms* ed J J Grefenstette (Hillsdale, NJ: Erlbaum)

Hancock P J B 1994 An empirical comparison of selection methods in evolutionary algorithms *Evolutionary Computing: AISB Workshop (Lecture Notes in Computer Science 865)* ed T C Fogarty (Berlin: Springer) pp 80–94

Harik G R and Goldberg D E 1996 Learning linkage *Foundations of Genetic Algorithms IV (Proc. Preprint)* pp 270–85

Hillis W D 1990 Co-evolving parasites improve simulated evolution as an optimisation procedure *Physica* **42D** 228–34

Holland J H 1975 *Adaptation in Natural and Artificial Systems* (Ann Arbor, MI: University of Michigan Press)

——1992 Genetic algorithms *Sci. Am.* **267** 66–72

Michalewicz Z 1992 *Genetic Algorithms + Data Structures = Evolution Programs* (Berlin: Springer)

Mitchell M, Forrest S and Holland J 1992 The royal road for genetic algorithms: fitness landscapes and GA performance *Proc. 1st Eur. Conf. on Artificial Life* (Cambridge, MA: MIT Press–Bradford)

Mühlenbein H 1992 How genetic algorithms really work. Part I: Mutation and hillclimbing *Parallel Problem Solving from Nature, 2* ed R Männer and B Manderick (Amsterdam: Elsevier–North-Holland)

Nix A and Vose M D 1991 Modeling genetic algorithms with Markov chains *Ann. Math. Artificial Intell.* **5** 79–88

Radcliffe N J 1990 *Genetic Neural Networks on MIMD Computers* PhD Thesis, University of Edinburgh

——1991 Forma analysis and random respectful recombination *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, 1991)* ed R K Belew and L B Booker (San Mateo, CA: Morgan Kaufmann) pp 222–9

Radcliffe N J and Surry P D 1994 Fitness variance of formae and performance prediction *Foundations of Genetic Algorithms III* ed L D Whitley and M D Vose (San Mateo, CA: Morgan Kaufmann) pp 51–72

——1995 Fundamental limitations on search algorithms: evolutionary computing in perspective *Computer Science Today: Recent Trends and Developments (Lecture Notes in Computer Science 1000)* ed J van Leeuwen (New York: Springer) pp 275–91

Schwefel H-P 1995 *Evolution and Optimum Seeking* (New York: Wiley)

Spears W M and De Jong K A 1991 On the virtues of parameterised uniform crossover *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, 1991)* ed R K Belew and L B Booker (San Mateo, CA: Morgan Kaufmann) pp 230–6

Syswerda G 1989 Uniform crossover in genetic algorithms *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, WA, 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann)

Vose M D 1992 Modelling simple genetic algorithms *Foundations of Genetic Algorithms 2* ed D Whitley (San Mateo, CA: Morgan Kaufmann)

Vose M D and Liepins G E 1991a Punctuated equilibria in genetic search *Complex Syst.* **5** 31–44

——1991b Schema disruption *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, 1991)* ed R K Belew and L B Booker (San Mateo, CA: Morgan Kaufmann) pp 237–43

Weinholt W 1993 A refined genetic algorithm for parameter optimization problems *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 589–96

Wolpert D H and Macready W G 1995 *No Free Lunch Theorems for Search* Santa Fe Institute Technical Report SFI-TR-95-02-010

B2.6 Transform methods

Sami Khuri

Abstract

Theoretical analysis of fitness functions in genetic algorithms has included the use of Walsh functions, which form a convenient basis for the expansion of fitness functions. These orthogonal, rectangular functions, which take values of ± 1, are more practical as a basis than the traditional sine/cosine basis. Walsh functions have also been used to compute the average fitness values of schemata, to decide whether a certain function is hard or easy for a genetic algorithm, and to design deceptive functions for the genetic algorithm as described in the first part of this article. This section also explores the use of Haar functions for the same purposes and highlights the computational advantages that they have over Walsh functions.

B2.6.1 Walsh analysis and Walsh transforms

B2.6.1.1 Introduction

Traditionally, Fourier series and transforms have been used to represent large classes of functions by superpositioning sine and cosine functions. More recently, other classes of complete, orthogonal functions have been used for the same purpose (Harmuth 1968). These new functions are rectangular and are easier to define and use with digital logic. Walsh functions have been used for analyzing various natural events. Much of the information pertaining to such events is in the form of signals which are functions of time (Beauchamp 1984). These signals can be studied, classified and analyzed using orthogonal functions and transformations. Walsh functions form a set whose members are simple functions that are easy to generate and define. Discrete and continuous signals can be represented as combinations of members of Walsh functions. Only orthogonal functions can completely synthesize a given time function accurately. Orthogonal functions also possess many important mathematical properties which makes them highly useful as an analysis tool.

Walsh functions form a complete orthogonal set of functions and can thus be used as a basis. These rectangular functions, which take values ± 1, are more practical, as a basis, than the traditional trigonometric basis (Bethke 1980). They have been used as theoretical tools to compute the average fitness values of hyperplanes, to decide whether a certain function is hard or easy for a genetic algorithm, and to design deceptive functions for the genetic algorithm (Bethke 1980, Goldberg 1989a, b).

In this part, most of which is from Goldberg's articles (1989a, b), we explore the use of Walsh functions as bases for fitness functions of *genetic algorithms*. In the second part, B2.6.2, an alternative to Walsh functions, Haar functions, are investigated, followed by a comparison between the two. B1.2

B2.6.1.2 Walsh functions as bases

Any fitness function defined on *binary strings* of length ℓ can be represented as a linear combination of discrete Walsh functions. When working with Walsh functions, it is more convenient to have strings with ± 1 rather than 0 and 1. The *auxiliary function* (Goldberg 1989a) associates with each binary string $x = x_\ell x_{\ell-1} \ldots x_2 x_1$ an auxiliary string $y = y_\ell y_{\ell-1} \ldots y_2 y_1$ where $y_i = 1 - 2x_i$ for $i = 1, 2, \ldots, \ell$ or C1.2

equivalently $x_i = \frac{1}{2}(1 - y_i)$. Hence $y_i \in \{+1, -1\}$ for all $i = 1, \ldots, \ell$. Given a string x, the auxiliary string y is defined as $y = \text{aux}(x_\ell)\text{aux}(x_{\ell-1}) \ldots \text{aux}(x_2)\text{aux}(x_1)$ where

$$y_i = \text{aux}(x_i) = \begin{cases} -1 & \text{if } x_i = 1 \\ 1 & \text{if } x_i = 0. \end{cases}$$

The *Walsh functions* (monomials in Goldberg 1989a) over auxiliary string variables form a set of 2^ℓ monomials defined for $y = \text{aux}(x)$:

$$\Psi_j(y) = \prod_{i=1}^{\ell} y_i^{j_i}$$

where $j = j_\ell j_{\ell-1} \ldots j_1$ and $j = \sum_{i=1}^{\ell} j_i 2^{i-1}$.

Example. Let us compute the monomial $\Psi_3(y)$ where $\ell = 3$. Since $j = 3$, then $j_3 = 0$, $j_2 = 1$ and $j_1 = 1$. Therefore $\Psi_3(y) = y_3^0 y_2^1 y_1^1 = y_1 y_2$. In other words, $j = 3$ signals the presence of positions one and two, which correspond to the indices of y ($\Psi_3(y) = y_1 y_2$). This ease of conversion between the indices and the Walsh monomials is one of the reasons behind the success of Walsh functions as a basis. If $x = 101$ then we may write

$$\begin{aligned} \Psi_3(\text{aux}(101)) &= \Psi_3(\text{aux}(1)\text{aux}(0)\text{aux}(1)) \\ &= \Psi_3(-11-1) \\ &= (-1)(1) \\ &= -1. \end{aligned}$$

The set $\{\Psi_j(y) : j = 0, 1, 2, \ldots, 2^\ell - 1\}$ forms a basis for the fitness functions defined on $[0, 2^\ell)$. That is

$$f(x) = \sum_{j=0}^{2^\ell - 1} w_j \Psi_j(x) \tag{B2.6.1}$$

where the w_j are the Walsh coefficients, $\Psi_j(x)$ are the Walsh monomials, and by $\Psi_j(x)$ we mean $\Psi_j(\text{aux}(x))$. The Walsh coefficients are given by

$$w_j = \frac{1}{2^\ell} \sum_{x=0}^{2^\ell - 1} f(x)\Psi_j(x). \tag{B2.6.2}$$

The subscript j of the Walsh monomial Ψ_j denotes the index. The number of ones in the binary expansion of j is the weight of the index. Thus, $\Psi_3(x)$ has three as index and is of weight two.

Since $\Psi_j(x) \neq 0$ for each $j \in [0, 2^\ell)$, unless $f(x)$ is orthogonal to a Walsh function $\Psi_j(x)$, the expansion of $f(x)$ as a linear combination of Walsh monomials has 2^ℓ nonzero terms. Thus, at most 2^ℓ nonzero terms are required for the expansion of a given function as a linear combination of Walsh functions. The total number of terms required for the computation of the expansion of the fitness function at a given point is $2^{2\ell}$.

$\Psi_j(x)$ is defined for discrete values of $x \in [0, 2^\ell)$. The function can be extended to obtain step functions (Beauchamp 1984) by allowing all values of t in the interval $[0, 2^\ell)$ and letting $\Psi_j(t) = \Psi_j(x_s)$ for $t \in [x_s, x_s + 1)$.

Table B2.6.1 gives the Walsh functions on $[0, 8)$.

The Walsh monomials presented here are in natural order, also known as Hadamard order. Other well-known orderings include the sequency order (Walsh 1923), the Rademacher order (Alexits 1961), the Harmuth order (Harmuth 1972), and the Boolean synthesis order (Gibbs 1970).

The natural order is known as Hadamard because the Walsh coefficients can be obtained using the Hadamard matrix. The rows and columns of the Hadamard matrix are orthogonal to one another. The lowest-order Hadamard matrix is

$$H_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}.$$

Table B2.6.1. Walsh functions on $[0, 8)$.

x	$\Psi_0(x)$	$\Psi_1(x)$	$\Psi_2(x)$	$\Psi_3(x)$	$\Psi_4(x)$	$\Psi_5(x)$	$\Psi_6(x)$	$\Psi_7(x)$
000	1	1	1	1	1	1	1	1
001	1	−1	1	−1	1	−1	1	−1
010	1	1	−1	−1	1	1	−1	−1
011	1	−1	−1	1	1	−1	−1	1
100	1	1	1	1	−1	−1	−1	−1
101	1	−1	1	−1	−1	1	−1	1
110	1	1	−1	−1	−1	−1	1	1
111	1	−1	−1	1	−1	1	1	−1
index j	0	1	2	3	4	5	6	7
$j_3 j_2 j_1$	000	001	010	011	100	101	110	111
monomials	$y_3^0 y_2^0 y_1^0$	$y_3^0 y_2^0 y_1^1$	$y_3^0 y_2^1 y_1^0$	$y_3^0 y_2^1 y_1^1$	$y_3^1 y_2^0 y_1^0$	$y_3^1 y_2^0 y_1^1$	$y_3^1 y_2^1 y_1^0$	$y_3^1 y_2^1 y_1^1$
$y_3^{j_3} y_2^{j_2} y_1^{j_1}$	1	y_1	y_2	$y_1 y_2$	y_3	$y_1 y_3$	$y_2 y_3$	$y_1 y_2 y_3$

The following recursive relation generates higher-order Hadamard matrices of order N:

$$H_N = H_{N/2} \otimes H_2$$

where $\otimes$ is the Kronecker or direct product. The Kronecker product in the above equation consists in replacing each element in H_2 by $H_{N/2}$; that is, 1 elements are replaced by H_2 and -1 elements by $-H_2$. This is called the Sylvester construction (MacWilliams and Sloane 1977), and thus

$$H_N = \begin{bmatrix} H_{N/2} & H_{N/2} \\ H_{N/2} & -H_{N/2} \end{bmatrix}.$$

For instance, H_8 is the following matrix:

$$H_8 = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{bmatrix}.$$

The Walsh coefficients can be obtained by the following equality:

$$\begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_{2^\ell - 1} \end{bmatrix} = \frac{1}{2^\ell} H_{2^\ell} \begin{bmatrix} f(x_0) \\ f(x_1) \\ \vdots \\ f(x_{2^\ell - 1}) \end{bmatrix}.$$

The second factor behind Walsh functions' popularity in genetic algorithms is the relatively small number of terms required in the expansion of schemata as linear combinations of Walsh coefficients.

B2.6.1.3 Schema average fitness

The average fitness value of a schema can be expressed as a partial signed summation over Walsh coefficients. Recall that a schema h (or hyperplane in the search space) is of the form $h_\ell, h_{\ell-1}, \ldots, h_2, h_1$ where $h_i \in \{0, 1, \star\}$ for all $i = 1, \ldots, \ell$, where $\star$ is either 0 or 1.

The schema average fitness values can be written in terms of Walsh coefficients. If $|h|$ denotes the number of elements in schema h, then the schema average fitness, $f(h)$, is given by Bethke (1980) and Goldberg (1989a):

$$f(h) = \frac{1}{|h|} \sum_{x \in h} f(x).$$

By substituting the value of $f(x)$, given by equation (B2.6.1), and by rearranging the summations, we obtain

$$f(h) = \frac{1}{|h|} \sum_{x \in h} \sum_{j=0}^{2^\ell - 1} w_j \Psi_j(x)$$
$$= \frac{1}{|h|} \sum_{j=0}^{2^\ell - 1} w_j \sum_{x \in h} \Psi_j(x)$$

where by $\Psi_j(x)$ we mean $\Psi_j(\mathrm{aux}(x))$. Expressing the schema (rather than the individual points of the search space) as a linear combination of Walsh functions has a very advantageous effect: the lower the order of the schema, the fewer the number of terms in the summation. As can be seen in table B2.6.2, in which six schemata of length three are written as linear combinations of the Walsh coefficients, the schemata with low order, such as $\star\star 0$, need only two terms, while 001 is of higher order and has eight terms in its expansion.

Table B2.6.2. Fitness averages as partial Walsh sums of some schemata.

Schema	Fitness averages
$\star\star\star$	w_1
$\star\star 0$	$w_0 + w_1$
$1\star\star$	$w_0 - w_4$
$\star 01$	$w_0 - w_1 + w_2 - w_3$
$11\star$	$w_0 - w_2 - w_4 + w_6$
001	$w_0 - w_1 + w_2 - w_3 + w_4 - w_5 + w_6 - w_7$

B2.6.1.4 Easy and hard problems for genetic algorithms

In his quest to define functions that are suitable for genetic algorithms, Goldberg, using Walsh coefficients, first designed minimal deceptive functions of two types, and later constructed fully deceptive three-bit functions. If 111 is the optimal point in the search space, then all order-one schemata lead away from it. In other words, we have $f(\star\star 1) < f(\star\star 0)$, $f(\star 1\star) < f(\star 0\star)$, and $f(1\star\star) < f(0\star\star)$.

Similarly, for order-two schemata, we want $f(\star 00) > f(\star 01)$, $f(\star 00) > f(\star 10)$, and $f(\star 00) > f(\star 11)$. By using the methods described in the previous section, the above inequalities are cast in terms of Walsh coefficients, and the problem reduces to solving a simultaneous set of inequalities. Many such fully deceptive functions can be designed.

Goldberg (1989b) also used neighborhood arguments and devised ANODE: analysis of deception, an algorithm for determining whether and to what degree a coding function combination is deceptive. The algorithm is applied to an instance of a problem, and decides whether or not that particular instance is deceptive for the genetic algorithm. In other words, the diagnostic is problem instance dependent, unlike other, more general theories that associate labels with problems (and not instances), such as tractability issues and the classification of problems into the classes of P- and NP-complete, for instance.

It is also crucial to keep in mind that the above analysis is of static nature, and that simple genetic algorithms were not designed to be optimization algorithms for static optimization problems in the first place (DeJong 1992). For a different kind of analysis with a more dynamic flavor, the reader is referred to the work of Fogel (1992) and Rudolph (1994), for instance, in which *Markov chains* are used to model genetic algorithms and to tackle conditions under which global convergence is achieved. B2.2.2

B2.6.1.5 Fast Walsh transforms

By taking advantage of the many repetitive computations performed with orthogonal transformations, the analysis can be implemented in the order of at most $N \log_2 N$ computations to form *fast transforms*. Note that $N = 2^\ell$. Modeled after the fast Fourier transform (Cooley and Tukey 1965), several fast Walsh transforms have been proposed (Shanks 1969). Since memory storage for intermediate-stage

computations is not needed, these are in-place algorithms. The calculated pair of output values can replace the corresponding pair of data in the preceding stage.

B2.6.1.6 Conclusion

In summary, Walsh functions form convenient bases. They can thus be used as practical transforms for discrete objective functions in optimization problems such as fitness functions in genetic algorithms. They are used to calculate the average fitness value of a schema, to decide whether a certain function is hard or easy for a genetic algorithm, and to design deceptive functions for the genetic algorithm.

B2.6.2 Haar analysis and Haar transforms

B2.6.2.1 Introduction

Walsh functions, which take values ± 1, form a complete orthogonal set of functions and can be used as a basis, to calculate the average fitness value of a schema, to decide whether a certain function is hard or easy for a genetic algorithm, and to design deceptive functions for the genetic algorithm (Bethke 1980, Goldberg 1989a, b). Any fitness function defined on binary strings of length ℓ can be represented as a linear combination of discrete Walsh functions.

If $\Psi_j(x)$ denotes a Walsh function, then $\{\Psi_j(x) : j = 0, 1, 2, \ldots, 2^\ell - 1\}$ forms a basis for the fitness functions defined on $[0, 2^\ell)$, and

$$f(x) = \sum_{j=0}^{2^\ell - 1} w_j \Psi_j(x) \tag{B2.6.3}$$

where the w_j are the Walsh coefficients given by

$$w_j = \frac{1}{2^\ell} \sum_{x=0}^{2^\ell - 1} f(x)\Psi_j(x). \tag{B2.6.4}$$

This part explores the use of another orthogonal, rectangular function: Haar functions (Haar 1910), that can be used as a convenient basis for the expansion of fitness functions. Haar functions can be processed more efficiently than Walsh functions: if ℓ denotes the size of each binary string in the solution space, at most 2^ℓ nonzero terms are required for the expansion of a given function as a linear combination of Walsh functions, while at most $\ell + 1$ nonzero terms are required with Haar expansion. Similarly, Haar coefficients require less computation than their Walsh counterparts. The total number of terms required for the computation of the expansion of the fitness function at a given point using Haar is of order 2^ℓ, which is for large ℓ substantially less than Walsh's $2^{2\ell}$ and the advantage of Haar over Walsh functions is of order $\ell\, 2^\ell$ when fast transforms are used.

B2.6.2.2 Haar functions

The set of Haar functions also forms a complete set of orthogonal rectangular basis functions. These functions were proposed by the Hungarian mathematician Haar (1910), approximately 13 years before Walsh's work. Haar functions take values of 1, 0, and -1, multiplied by powers of $2^{1/2}$. The interval Haar functions are defined on is usually normalized to $[0, 1)$ (see e.g. Kremer 1973).

One could also use the unnormalized Haar functions, taking values of 0 and ± 1 (see e.g. Khuri 1994). In the following definition, the Haar functions are defined on $[0, 2^\ell)$, and are not normalized,

$$H_0(x) = 1 \qquad \text{for } 0 \le x < 2^\ell$$

$$H_1(x) = \begin{cases} 1 & \text{for } 0 \le x < 2^{\ell-1} \\ -1 & \text{for } 2^{\ell-1} \le x < 2^\ell \end{cases}$$

$$H_2(x) = \begin{cases} 2^{1/2} & \text{for } 0 \le x < 2^{\ell-2} \\ -2^{1/2} & \text{for } 2^{\ell-2} \le x < 2^{\ell-1} \\ 0 & \text{elsewhere in } [0, 2^\ell) \end{cases}$$

$$H_3(x) = \begin{cases} 0 & \text{for } 0 \le x < 2^{\ell-1} \\ 2^{1/2} & \text{for } (2)2^{\ell-2} \le x < (3)2^{\ell-2} \\ -2^{1/2} & \text{for } (3)2^{\ell-2} \le x < (4)2^{\ell-2} \end{cases}$$

$$\vdots$$

$$H_{2^q+m}(x) = \begin{cases} (2^{1/2})^q & \text{for } (2m)2^{\ell-q-1} \le x < (2m+1)2^{\ell-q-1} \\ -(2^{1/2})^q & \text{for } (2m+1)2^{\ell-q-1} \le x < (2m+2)2^{\ell-q-1} \\ 0 & \text{elsewhere in } [0, 2^\ell) \end{cases} \tag{B2.6.5}$$

$$\vdots$$

$$H_{2^\ell-1}(x) = \begin{cases} (2^{1/2})^{\ell-1} & \text{for } 2(2^{\ell-1}-1) \le x < 2^\ell - 1 \\ -(2^{1/2})^{\ell-1} & \text{for } 2^\ell - 1 \le x < 2^\ell \\ 0 & \text{elsewhere in } [0, 2^\ell). \end{cases}$$

For every value of $q = 0, 1, \ldots, \ell - 1$, we have $m = 0, 1, \ldots, 2^q - 1$.

Table B2.6.3 shows the set of eight Haar functions for $\ell = 3$. The Haar function, $H_{2^q+m}(x)$, has degree q and order m. Functions with the same degree are translations of each other.

The set $\{H_j(x) : j = 0, 1, 2, \ldots, 2^\ell - 1\}$ forms a basis for the fitness functions defined on the integers in $[0, 2^\ell)$. That is

$$f(x) = \sum_{j=0}^{2^\ell-1} h_j \, H_j(x) \tag{B2.6.6}$$

where the h_j, for $j = 2^q + m$, are the Haar coefficients given by

$$h_j = \frac{1}{2^\ell} \sum_{x=0}^{2^\ell-1} f(x) \, H_j(x). \tag{B2.6.7}$$

Table B2.6.3. Haar functions $H_{2^q+m}(x)$ for $\ell = 3$.

x	$H_0(x)$	$H_1(x)$	$H_2(x)$	$H_3(x)$	$H_4(x)$	$H_5(x)$	$H_6(x)$	$H_7(x)$
000	1	1	$2^{1/2}$	0	2	0	0	0
001	1	1	$2^{1/2}$	0	-2	0	0	0
010	1	1	$-2^{1/2}$	0	0	2	0	0
011	1	1	$-2^{1/2}$	0	0	-2	0	0
100	1	-1	0	$2^{1/2}$	0	0	2	0
101	1	-1	0	$2^{1/2}$	0	0	-2	0
110	1	-1	0	$-2^{1/2}$	0	0	0	2
111	1	-1	0	$-2^{1/2}$	0	0	0	-2
index j	0	1	2	3	4	5	6	7
degree q	undefined	0	1	1	2	2	2	2
order m	undefined	0	0	1	0	1	2	3

As equation (B2.6.5) and table B2.6.3 indicate, the higher the degree q, the smaller the subinterval with nonzero values for $H_j(x)$. Consequently, each Haar coefficient depends only on the local behavior of $f(x)$.

More precisely, from its definition (see equation (B2.6.5)), we have that $H_{2^q+m}(x) \neq 0$ only for $m2^{\ell-q} \le x < (m+1)2^{\ell-q}$. Every degree q partitions the interval $[0, 2^\ell)$ into pairwise disjoint subintervals: $[0, 2^{\ell-q})$, $[2^{\ell-q}, (2)2^{\ell-q})$, $[(2)2^{\ell-q}, (3)2^{\ell-q})$, …, $[(2^q-1)(2^{\ell-q}), 2^q(2^{\ell-q}))$, each of width $2^{\ell-q}$ and such that $H_{2^q+m}(x) = 0$ on all but one of the subintervals.

The search space contains 2^ℓ points and each subinterval will have $2^{\ell-q}$ points x such that $H_{2^q+m}(x) \neq 0$. Thus, by the definition of h_{2^q+m} (B2.6.7), there are at most $2^{\ell-q}$ nonzero terms in the computation. The following results are equivalent to Beauchamp's concerning the linear combination of the Haar coefficients h_{2^q+m} where $m < 2^q$ (Beauchamp 1984).

Result B2.6.1. Every Haar coefficient of degree q has at most $2^{\ell-q}$ nonzero terms. Each term corresponds to a point in an interval of the form $[(i)2^{\ell-q}, (i+1)2^{\ell-q})$. Consequently, the linear combination of each Haar coefficient h_j, where $j = 2^q + m$, has at most $2^{\ell-q}$ nonzero terms. In addition, h_0 has at most 2^ℓ nonzero terms. □

A similar result holds for the computation of $f(x)$ in equation (B2.6.6). In the linear combination, for a given x, only a few terms have nonzero values. Since $H_0(x) \neq 0$ and $H_1(x) \neq 0$ for all $x \in [0, 2^\ell)$, $H_0(x)$ and $H_1(x)$ appear in the right-hand side of equation (B2.6.7) for any given x. We have already seen that degree $q > 0$ partitions $[0, 2^\ell)$ into $2^{\ell-q}$ pairwise disjoint subintervals: $[0, 2^{\ell-q})$, $[2^{\ell-q}, (2)2^{\ell-q})$, $[(2)2^{\ell-q}, (3)2^{\ell-q})$, $\ldots$, $[(2^q-1)(2^{\ell-q}), 2^q(2^{\ell-q}))$, each of width $2^{\ell-q}$ and such that $H_{2^q+m}(x) = 0$ except on the subinterval $[(m)2^{\ell-q}, (m+1)2^{\ell-q})$ for $m = 0, 1, \ldots, 2^q - 1$. Hence, for a given $x \in [0, 2^\ell)$, and a given q, $H_{2^q+m}(x)$ is nonzero for $m = i$, and zero for all other values of m. Thus, each degree q contributes at most one nonzero Haar function in the right-hand side of equation (B2.6.6), which can be rewritten as

$$f(x) = h_0 H_0(x) + h_1 H_1(x) + \sum_{q=1}^{\ell-1} \sum_{m=0}^{2^q-1} h_{2^q+m} H_{2^q+m}(x). \tag{B2.6.8}$$

For each degree q, $\sum_{m=0}^{2^q-1} h_{2^q+m} H_{2^q+m}(x)$ has at most one nonzero term. From equation (B2.6.8), the total number of nonzero terms is at most $2 + (\ell - 1) = \ell + 1$. We have shown the following result (Karpovsky 1985).

Result B2.6.2. For any fixed value $x \in [0, 2^\ell)$, $f(x)$ has at most $\ell + 1$ nonzero terms. □

According to result B2.6.1, every Haar coefficient of degree q, $q > 1$, has at most $2^{\ell-q}$ nonzero terms in its computation (equation (B2.6.7)). Since, however, Walsh functions are never zero, each Walsh coefficient can be written as a linear combination of at most 2^ℓ nonzero terms (see equation (B2.6.4)). According to result B2.6.2, for any fixed value x, $f(x) = \sum_{j=0}^{2^\ell-1} h_j H_j(x)$, has at most $\ell+1$ terms. Again, since Walsh functions are never zero, at most 2^ℓ nonzero terms are required for the Walsh expansion (see equation (B2.6.3)). These results are illustrated by considering a simple problem instance of the integer knapsack problem.

Example B2.6.1. The integer *knapsack problem* consists of a knapsack of capacity M, and of ℓ objects with associated weights $w_1, \ldots, w_\ell$, and profits $p_1, \ldots, p_\ell$. The knapsack is to be filled with some of the objects without exceeding M, and such as to maximize the sum of the profits of the selected objects. In other words: G9.7

Maximize $\sum_{i \in S} p_i$ where $S \subseteq \{1, 2, \ldots, \ell\}$ and $\sum_{i \in S} w_i \leq M$.

The solution space can be encoded as 2^ℓ binary strings $x_\ell x_{\ell-1} \ldots x_1$ where $x_i = 1$ means that object i is placed in the knapsack and $x_i = 0$ means that object i is not selected. Each string $\boldsymbol{x} = x_\ell \ldots x_1$, where $x_i \in \{0, 1\}$, and $1 \leq i \leq \ell$, has a profit associated with it: $P(\boldsymbol{x}) = \sum_{i=1}^{\ell} x_i\ p_i$. Some strings represent infeasible solutions. The weight of the solution is given by $W(\boldsymbol{x}) = \sum_{i=1}^{\ell} x_i\ w_i$. Infeasible strings are those whose weight $W(\boldsymbol{x})$ is greater than M.

The fitness $f(\boldsymbol{x})$ of strings is defined as follows: $f(\boldsymbol{x}) = P(\boldsymbol{x}) - \text{penalty}(\boldsymbol{x})$ where $\text{penalty}(\boldsymbol{x}) = 0$ if $\boldsymbol{x}$ is feasible. The penalty is a function of the weight and profit of the string, and of the knapsack capacity (Khuri and Batarekh 1990).

We now consider the following problem instance:

Objects:	4	3	2	1	
Weights:	10	5	8	9	and $M = 23$.
Profits:	15	8	10	6	

Table B2.6.4 gives the fitnesses of the strings $\boldsymbol{x} = x_4 x_3 x_2 x_1$ after a penalty has been applied to the infeasible strings (strings 1011, 1101, and 1111). The Walsh and Haar coefficients are given in table B2.6.5.

Table B2.6.4. Fitness values for problem instance.

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$f(x)$	0	6	10	16	8	14	18	24	15	21	25	12	23	10	33	3

Table B2.6.5. Walsh and Haar coefficients for the problem instance.

	$j=0$	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$	$j=6$	$j=7$
w_j	$\frac{238}{16}$	$\frac{26}{16}$	$\frac{-44}{16}$	$\frac{-36}{16}$	$\frac{-28}{16}$	$\frac{-36}{16}$	$\frac{2}{16}$	$\frac{-2}{16}$
h_j	$\frac{238}{16}$	$\frac{-46}{16}$	$\frac{-32}{16}2^{1/2}$	$\frac{4}{16}2^{1/2}$	$\frac{-40}{16}$	$\frac{-40}{16}$	$\frac{-2}{16}$	$\frac{-6}{16}$
	$j=8$	$j=9$	$j=10$	$j=11$	$j=12$	$j=13$	$j=14$	$j=15$
w_j	$\frac{-46}{16}$	$\frac{-74}{16}$	$\frac{-36}{16}$	$\frac{36}{16}$	$\frac{-36}{16}$	$\frac{36}{16}$	$\frac{-2}{16}$	$\frac{2}{16}$
h_j	$\frac{-12}{16}2^{1/2}$	$\frac{-12}{16}2^{1/2}$	$\frac{-12}{16}2^{1/2}$	$\frac{-12}{16}2^{1/2}$	$\frac{-12}{16}2^{1/2}$	$\frac{26}{16}2^{1/2}$	$\frac{26}{16}2^{1/2}$	$\frac{60}{16}2^{1/2}$

Note that the computation of w_{13}, for instance, has 16 terms (see equation (B2.6.4)) requiring the values of $f(x)$ at all points in the interval $[0, 16)$.

$$\begin{aligned} w_{13} &= \tfrac{1}{16}[0-6+10-16-8+14-18+24-15+21-25+12+23-10+3-3] \\ &= \tfrac{36}{16}. \end{aligned}$$

On the other hand (see result B2.6.1 where $q=3$ and $m=4$ since $13=2^3+4$), h_{13} requires the values of $f(x)$ at the two points $x=1010$ and $x=1011$ only, since $H_{13}=0$ for all other values of $x \in [0, 16)$.

$$\begin{aligned} h_{13} &= \tfrac{1}{16}[2(2^{1/2})(25)-2(2^{1/2})(12)] \\ &= \tfrac{26}{16}2^{1/2}. \end{aligned}$$

Similarly, $f(11)$ (i.e. $x=1011$) will require the computation of sixteen Walsh terms (see equation (B2.6.3)) instead of just five Haar terms (see result 2).

$$\begin{aligned} f(1011) &= \sum_{j=0}^{15} w_j\, \Psi_j(1011) \\ &= w_0 - w_1 - w_2 + w_3 + w_4 - w_5 - w_6 + w_7 - w_8 + w_9 + w_{10} - w_{11} - w_{12} + w_{13} + w_{14} - w_{15} \\ &= \tfrac{1}{16}[238-26+44-36-28+36-2-2+46-74-36-36+36+36-2-2] \\ &= 12 \\ f(1011) &= \sum_{j=0}^{15} h_j\, H_j(1011) \\ &= h_0 - h_1 + 2(2^{1/2})h_3 - 2h_6 - 2(2^{1/2})h_{13} \\ &= \tfrac{1}{16}[238+46+8+4-104] \\ &= 12. \end{aligned}$$

Note that the total number of terms required for the computation of the expansion of $f(1011)$ in the case of Walsh is 256 (16×16), and for Haar, 46 (16 each for h_0 and h_1; eight for h_3, four for h_6, and two terms for h_{13}). Thus, the computation of 210 more terms is required with Walsh than with Haar. In practical cases, ℓ is substantially larger than four. For instance, for $\ell = 20$ there are about $2^{40} \approx 10^{12}$ more terms using Walsh expansion. □

No comparison between Walsh and Haar would be complete without considering fitness averages of schemata (Goldberg 1989a).

A comparison between the maximum number of nonzero terms, and the total number of terms for the computation of all 81 schemata of length $\ell = 4$ is tabulated in table B2.6.6. A fixed position is represented with 'd' while '*' stands for a 'don't care'.

Consider, for example, the row in table B2.6.6 corresponding to d**d. It represents four schemata, $E = \{0**0, 0**1, 1**0, 1**1\}$. The average fitness of each one can be expressed as a linear combination of at most four nonzero Walsh coefficients. For instance,

$$f(1**0) = w_0 + w_1 - w_8 - w_9.$$

Table B2.6.6. Computing schemata for $\ell = 4$ with Walsh and Haar functions. (Reproduced from Khuri 1994.)

	Nonzero terms		Total number of nonzero terms	
Schema	Walsh	Haar	Walsh	Haar
****	1	1	16	16
***d	4	20	64	64
**d*	4	10	64	64
*d**	4	6	64	64
d***	4	4	64	64
**dd	16	36	256	160
*d*d	16	28	256	160
d**d	16	24	256	160
dd	16	20	256	160
d*d*	16	16	256	160
dd**	16	12	256	160
*ddd	64	56	1024	352
d*dd	64	48	1024	352
dd*d	64	40	1024	352
ddd*	64	32	1024	352
dddd	128	80	2048	736
Total	469	433	7952	3376

Since E has four schemata, the maximum number of nonzero terms for all schemata represented by d**d is $4 \times 4 = 16$ and is tabulated in the second column of table B2.6.6. Moreover, each single Walsh coefficient requires 16 terms for its computation (see equation (B2.6.4)). Thus the total number of terms required in the computation of the expansion of $f(1{**}0)$ is 4×16; and that of all schemata represented by d**d, 4×64, reported in the fourth column. On the other hand, the average fitness of each schema in E can be expressed as a linear combination of at most six nonzero Haar coefficients. For instance,

$$f(1{**}0) = h_0 - h_1 + \tfrac{1}{4}(h_{12} + h_{13} + h_{14} + h_{15}). \tag{B2.6.9}$$

The third column's entry has therefore the value 4×6. It might thus appear easier to use Walsh functions for this fitness average. Nevertheless, according to result B2.6.1, only two terms in equation (B2.6.7) are required for the computation of each of h_{12}, h_{13}, h_{14}, and h_{15}, while 16 are needed for h_0 and 16 for h_1. Likewise, it can be shown that 40 terms are required in the computation of the expansion of the other three schemata in E, bringing the total to 4×40 as reported in the last column of table B2.6.6. As can be seen in the last row of table B2.6.6, a substantial saving can be achieved by using Haar instead of Walsh functions.

With respect to fast transforms, they can be implemented with on the order of at most $\ell\, 2^\ell$ computations in the case of fast Walsh transforms and on the order of 2^ℓ for the fast Haar transforms (Roeser and Jernigan 1982). With these implementations, the difference between the total number of terms required for the computation of the expansions of Walsh and of Haar still remains exponential in ℓ (of order $\ell\, 2^\ell$). The computation of a single term in fast transforms is built upon the values of previous levels: many more levels for Walsh than Haar. Thus, many more computations (of the order $\ell\, 2^\ell$) are required for the computation of a single Walsh term. Fast transforms are represented by layered flowcharts where an intermediate result at a certain stage is obtained by adding (or subtracting) two intermediate results from the previous layer. Thus, when dealing with fast transforms, it is more appropriate to count the number of operations (additions or subtractions) which is equivalent to counting the number of terms (Roeser and Jernigan 1982). It can be shown that exactly $\ell\, 2^\ell - 2^{\ell+1} + 2$ more operations are needed with Walsh than with Haar functions. For instance, for $\ell = 20$, one needs to perform 18 875 002 more operations when the Walsh fast transform is used instead of the Haar transform. We conclude this section by noting that the Haar transforms 'are the fastest linear transformations presently available' (Beauchamp 1984).

B2.6.2.3 Conclusion

This work highlights the computational advantages that Haar functions have over Walsh monomials. The former can thus be used as practical transforms for discrete objective functions in optimization problems. More precisely, the total number of terms required for the computation of the expansion of the fitness function $f(x)$ for a given x using Haar functions is of order 2^{ℓ} which is substantially less than Walsh's $2^{2\ell}$. Similarly, we have seen that while w_j depends on the behavior of $f(x)$ at all 2^{ℓ} points, h_j depends only on the local behavior of $f(x)$ at a few points which are 'close together', and, furthermore, the advantage of Haar over Walsh functions remains very large (of order $\ell\, 2^{\ell}$) if fast transforms are used. One more advantage Haar functions have over Walsh functions is evident when they are used to approximate continuous functions. Walsh expansions might diverge at some points, whereas Haar expansions always converge (Alexits 1961).

References

Alexits G 1961 *Convergence Problems of Orthogonal Series* (New York: Pergamon)

Beauchamp K G 1984 *Applications of Walsh and Related Functions* (New York: Academic)

Bethke A D 1980 *Genetic Algorithms as Function Optimizers* Doctoral Dissertation, University of Michigan

Cooley J W and Tukey J W 1965 An algorithm for the machine calculation of complex Fourier series *Math. Comput.* **19** 297–301

De Jong K A 1992 Are genetic algorithms function optimizers? *Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature (Brussels, 1992)* ed R Männer and B Manderick (Amsterdam: Elsevier) pp 3–13

Fogel D B 1992 *Evolving Artificial Intelligence* Doctoral Dissertation, University of California at San Diego

Gibbs J E 1970 Discrete complex Walsh transforms *Proc. Symp. on Applications of Walsh Functions* pp 106–22

Goldberg D E 1989a Genetic algorithms and Walsh functions part I: a gentle introduction *Complex Syst.* **3** 129–52

——1989b Genetic algorithms and Walsh functions part II: deception and its analysis *Complex Syst.* **3** 153–71

Haar A 1910 Zur Theorie der orthogonalen Funktionensysteme *Math. Ann.* **69** 331–71

Harmuth H F 1968 A generalized concept of frequency and some applications *IEEE Trans. Information Theory* **IT-14** 375–82

——1972 *Transmission of Information by Orthogonal Functions* 2nd edn (Berlin: Springer)

Karpovsky M G 1985 *Spectral Techniques and Fault Detection* (New York: Academic)

Khuri S 1994 Walsh and Haar functions in genetic algorithms *Proc. 1994 ACM Symp. on Applied Computing* (New York: ACM) pp 201–5

Khuri S and Batarekh A 1990 Heuristics for the integer knapsack problem *Proc. 10th Int. Computer Science Conf. (Santiago)* pp 161–72

Kremer H 1973 On theory of fast Walsh transform algorithms *Colloq. on the Theory and Applications of Walsh and Other Non-Sinusoidal Functions (Hatfield Polytechnic, UK)*

MacWilliams F J and Sloane N J A 1977 *The Theory of Error-Correcting Codes* (New York: North-Holland)

Roeser P R and Jernigan M E 1982 Fast Haar transform algorithms *IEEE Trans. Comput.* **C-31** 175–7

Rudolph G 1994 Convergence analysis of canonical genetic algorithms *IEEE Trans. Neural Networks* **5** 96–101

Shanks J L 1969 Computation of the fast Walsh–Fourier transform *IEEE Trans. Comput.* **C-18** 457–9

Walsh J L 1923 A closed set of orthogonal functions *Ann. J. Math* **55** 5–24

B2.7 Fitness landscapes

Kalyanmoy Deb (B2.7.1), *Lee Altenberg* (B2.7.2), *Bernard Manderick* (B2.7.3), *Thomas Bäck* (B2.7.4), *Zbigniew Michalewicz* (B2.7.4), *Melanie Mitchell* (B2.7.5) *and Stephanie Forrest* (B2.7.5)

Abstract

See the individual abstracts for sections B2.7.1, B2.7.2, B2.7.3, B2.7.4 and B2.7.5.

B2.7.1 Deceptive landscapes

Kalyanmoy Deb

Abstract

In order to study the efficacy of evolutionary algorithms, a number of fitness landscapes have been designed and used as test functions. Since the optimal solution(s) of these fitness landscapes are known *a priori*, controlled experiments can be performed to investigate the convergence properties of evolutionary algorithms. A number of fitness landscapes are discussed in this section. These fitness landscapes are designed either to test some specific properties of the algorithms or to investigate overall working of the algorithms on difficult fitness landscapes.

B2.7.1.1 Introduction

Deceptive landscapes have been mainly studied in the context of *genetic algorithms* (GAs), although the concept of deceptive landscapes in creating difficult test functions can also be developed for other evolutionary algorithms. The development of deceptive functions lies in the proper understanding of the building block hypothesis. The building block hypothesis suggests that GAs work by combining low-order building blocks to form higher-order building blocks (see Section B2.5). Therefore, if in a function the low-order building blocks do not combine to form higher-order building blocks, GAs may have difficulty in optimizing the function. Deceptive functions are those functions where the low-order building blocks do not combine to form higher-order building blocks: instead they form building blocks for a suboptimal solution. The main motivation behind developing such functions is to create difficult test functions for comparing different implementations of GAs. It is then argued that if a GA succeeds in solving these difficult test functions, it can solve other simpler functions.

B1.2

B2.5

A deceptive function usually has at least two optimal solutions—one global and one local. A local optimal solution is the best solution in the neighborhood of the solution, whereas the global optimal solution is the best solution in the entire search space. Thus, the local solution is inferior to the global solution and is usually known as the deceptive solution. However, it has been shown elsewhere (Deb and Goldberg 1992, Whitley 1991) that the deceptive solution can be at most one-bit dissimilar to the local optimal solution in binary functions. A deceptive fitness function is designed by comparing the schemata representing the global optimal solution and the deceptive solution. The comparison is usually performed according to the fitness of the competing schemata. A deceptive function is designed by adjusting the

string fitness values in such a way that the schemata representing the deceptive solution have better fitness than any other schemata including that representing the global optimal solution in a schema partition. It is then argued that, because of the superiority of the deceptive schemata, GAs process them favorably in early generations. Solutions representing these schemata take over the population and GAs may finally find the deceptive solution, instead of the global optimal solution. Thus, the deceptive functions may cause GAs to find a suboptimal solution. Since these fitness landscapes are supposedly difficult for GAs, considerable effort has been spent in designing different deceptive functions and studies have been made to understand how simple GAs can be modified to solve such difficult landscapes (Deb 1991, Goldberg *et al* 1989, 1990). In the following, we first define deception and then outline simple procedures for creating deceptive functions.

B2.7.1.2 Schema deception

Although there exists some lack of agreement among researchers in the evolutionary computation (EC) community about the procedure of calculating the schema fitness and about the very definition of deception (Grefenstette 1993), we present here one version of the deception theory. Before we present that definition, two terminologies—schema fitness and schema partition—must be defined. A schema fitness is defined as the average fitness of all strings representing the schema. Thus, one schema is worse than another schema if the fitness of the former schema is inferior to that of the latter schema. A schema partition is represented by a binary string constructed with `f` and $*$, where a `f` represents a fixed position having either a `1` or a `0` (but not both) and a $*$ represents a 'don't care' symbol denoting either a `1` or a `0`. A schema partition represents 2^k schemata, where k is the number of fixed positions `f` in the partition. The parameter k is also known as the *order* of the schema partition. Thus, an order-k schema partition divides the entire search space into 2^k distinct and equal regions. For example, in a three-bit binary function, the second-order schema partition (`ff`$*$) represents four schemata: (`00`$*$), (`01`$*$), (`10`$*$), and (`11`$*$). Since each of these schemata represents two distinct strings, the schema partition (`ff`$*$) divides the entire search space into four equal regions. It is clear that a higher-order schema partition divides the search space into exponentially more regions than a lower-order schema partition. In the spirit of the above schema partition definition, it can be concluded that the highest-order (of order ℓ) schema partition divides the search space into 2^ℓ regions and each schema represents exactly one of the strings. Of course, the lowest-order (of order zero) schema partition has only one schema which represents all strings in the search space.

Definition B2.7.1. A schema partition is said to be deceptive if the schema containing the deceptive optimal solution is no worse than all other schemata in the partition.

We illustrate a deceptive schema partition in a three-bit function having its global and deceptive solutions at (`111`) and (`000`), respectively. According to the above definition of schema partition, the schema partition (`ff`$*$) is deceptive if the fitness of the schema (`00`$*$) is no worse than that of the other three schemata in the schema partition. For maximization problems, this requires the following three relationships to be true:

$$F(00*) \geq F(01*) \tag{B2.7.1}$$

$$F(00*) \geq F(10*) \tag{B2.7.2}$$

$$F(00*) \geq F(11*). \tag{B2.7.3}$$

Definition B2.7.2. A function is said to be fully deceptive if all $2^\ell - 2$ (see below) schema partitions are deceptive.

In an ℓ-bit problem, there are a total of 2^ℓ schema partitions, of which two of them (one with all fixed positions and the other with all $*$) cannot be deceptive. Thus, if all other $(2^\ell - 2)$ schema partitions are deceptive according to the above definition, the function is fully deceptive. Deb and Goldberg (1994) calculated that about $O(4^\ell)$ floating point operations are required to create a fully deceptive function. A function can also be partially deceptive to a certain order.

Definition B2.7.3. A function is said to be partially deceptive to order k if all schema partitions of order smaller than k are deceptive.

B2.7.1.3 Deceptive functions

Many researchers have created partially and fully deceptive functions from different considerations. Goldberg (1989a) created a three-bit fully deceptive function by explicitly calculating and comparing all schema fitness values. Liepins and Vose (1990) and Whitley (1991) have calculated different fully deceptive functions from intuition. Goldberg (1990) derived a fully deceptive function from low-order Walsh coefficients. Deb and Goldberg (1994) have created fully and partially deceptive trap functions (trap functions were originally introduced by Ackley (1987)) and found sufficient conditions to test and create deceptive functions. Since trap functions are piecewise linear functions and are defined with only a few independent function values, they are easy to manipulate and analyze to create a deceptive function. In the following, we present a fully deceptive trap function.

Trap functions are defined in terms of *unitation* (the number of 1s in a string). A function of unitation has the same function value for all strings of identical unitation. That is, in a three-bit unitation function, the strings (001), (010), and (100) have the same function value (because all the above three strings have the same unitation of one). Thus, in a ℓ-bit unitation function there are only $(\ell + 1)$ different function values. This reduction in number of function values (from 2^ℓ to $(\ell + 1)$) has helped researchers to create deceptive functions using unitation functions. A trap function $f(u)$, as a function of unitation u, is defined as follows (Ackley 1987):

$$f(u) = \begin{cases} \dfrac{a}{z}(z-u) & \text{if } u \le z \\ \dfrac{b}{\ell - z}(u-z) & \text{otherwise} \end{cases} \tag{B2.7.4}$$

where a and b are the function values of the deceptive and global optimal solutions, respectively. The trap function is a piecewise linear function that divides the search space into two basins in the unitation space, one leading to the global optimal solution and other leading to the deceptive solution. Figure B2.7.1 shows a trap function as a function of unitation (left) and as a function of the decoded value of the binary string (right) with $a = 0.6$, $b = 1.0$, and $z = 3$. The parameter z is the slope change location and u is

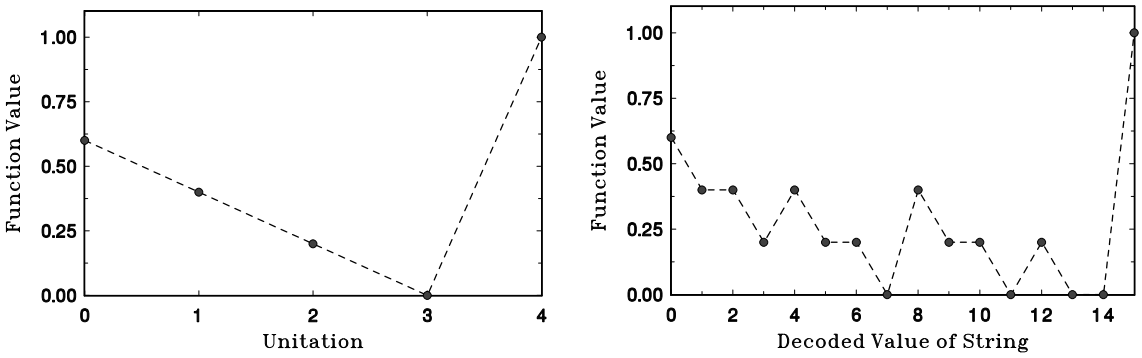


Figure B2.7.1. A four-bit trap function ($a = 0.6$, $b = 1.0$, and $z = 3$) as a function of unitation (left) and as a function of binary strings (right).

the unitation of a string. Deb and Goldberg (1994) have found that an ℓ-bit trap function becomes fully deceptive if the following condition is true (for small ℓ and $a \approx b$):

$$z \ge \frac{\ell}{2} + \frac{(b-a)}{(b+a)}\frac{\ell(\ell-1)}{2}. \tag{B2.7.5}$$

The above condition suggests that in a deceptive trap function the slope change location z is closer to ℓ. In other words, there are more strings in the basin of the deceptive solution than that in the basin of the global optimal solution. Using the above condition, we create a six-bit fully deceptive trap function with the strings (000000) and (111111) being the deceptive and global optimal solutions ($a = 0.92$, $b = 1.00$, and $z = 4$):

```
f(000000)=0.920  f(*00000)=0.805  f(**0000)=0.690  f(***000)=0.575  f(****00)=0.460  f(*****0)=0.367
f(000001)=0.690  f(*00001)=0.575  f(**0001)=0.460  f(***001)=0.345  f(****01)=0.275  f(*****1)=0.274
f(000011)=0.460  f(*00011)=0.345  f(**0011)=0.230  f(***011)=0.206  f(****11)=0.273
f(000111)=0.230  f(*00111)=0.115  f(**0111)=0.182  f(***111)=0.341
f(001111)=0.000  f(*01111)=0.250  f(**1111)=0.500
f(011111)=0.500  f(*11111)=0.750
f(111111)=1.000
```

The leftmost column shows seven different function values in a six-bit unitation function and other columns show the schema fitness values of different schema partitions. In the above function, the deceptive solution has a function value equal to 0.92 and the global solution has a function value equal to 1.00. The string (010100) has a function value equal to 0.460, because all strings of unitation 2 have a function value 0.460. In functions of unitation, all schema of a certain order and unitation also have the same fitness. That is, the schema (00∗010) has a fitness value equal to 0.575, because this schema is of order five and of unitation one, and all schema of order five and unitation one have a fitness equal to 0.575. The above schema fitness calculations show that the schema containing the deceptive solution is no worse than any other schemata in each partition. For example, for any schema partition of order two, the schema containing the deceptive solution has a fitness equal to 0.690, which is better than any other schema in that partition (third column). However, the deceptive string (000000) is not the true optimal solution. Thus, the above schema partition is deceptive. Since all $2^6 - 2$ or 62 schema partitions are deceptive, the above fitness landscape is fully deceptive.

Although in the above deceptive landscape the string of all 1s is considered to be the globally best string, any other string c can also be the globally best string. In this case, the above function values are assigned to another set of strings obtained by performing a bitwise exclusive-or operation to the above strings with the complement of c (Goldberg 1990).

B2.7.1.4 Sufficient conditions for deception

Deb and Goldberg (1994) have also found sufficient conditions for any arbitrary function to be fully deceptive (assuming that the strings of all 1s and all 0s are global and deceptive solutions, respectively):

$$\begin{array}{ll} \text{primary optimality condition:} & f(\ell) > \max[f(0), \max f(1)] \\ \text{primary deception condition:} & f(0) > \max[\max f(2), (f(\ell) - (\min f(1) - \max f(\ell - 1))] \\ \text{secondary deception condition:} & \min f(i) \geq \max f(j) \quad \text{for } 1 \leq i \leq \lfloor \ell/2 \rfloor \text{ and } i < j \leq \ell - i \end{array} \tag{B2.7.6}$$

where $\min f(i)$ and $\max f(i)$ are the minimum and maximum function values of all strings having a unitation i. A fitness function satisfying the above conditions is guaranteed to be a fully deceptive function; however a function not satisfying any of the above conditions may also be deceptive. However, Deb and Goldberg (1994) have observed that the above conditions can prove deception in most of the deceptive functions that exist in the GA literature. These sufficient conditions allow a systematic way of creating a deceptive function and a quick way to test deception in any arbitrary function. The number of floating-point operations required to design a fully deceptive function using the above conditions is only $O(\ell^2)$, whereas $O(4^\ell)$ operations are required to create a deceptive function with the consideration of all schema partition deception.

B2.7.1.5 Other deceptive functions

Goldberg *et al* (1992) have also defined multimodal deceptive functions and developed a method to create fully or partially deceptive multimodal functions from low-order Walsh coefficients. Mason (1991) has developed a method to create deceptive functions for nonbinary functions. Kargupta *et al* (1992) have also suggested a method to create deceptive problems in *permutation problems*. C1.4

The design of deceptive landscapes and subsequent attempts to solve such functions have provided better insights into the working of GAs and helped to develop modified GAs to solve such difficult functions. The *messy GA* (Deb 1991, Goldberg *et al* 1989, 1990) is a derivative of such considerations C4.2.4 and has been used to solve massively multimodal, deceptive, and highly nonlinear functions in only $O(\ell \log \ell)$ function evaluations, where ℓ is the number of binary variables (Goldberg *et al* 1993). These results are remarkable and set up standards for other competitive algorithms to achieve, but what is yet

more remarkable is the development of such efficient algorithms through proper understanding of the complex mechanisms of GAs and their extensions for handling difficult fitness landscapes.

B2.7.2 NK landscapes

Lee Altenberg†

Abstract

NK fitness landscapes are stochastically generated fitness functions on bit strings, parameterized (with N genes and K interactions between genes) so as to make them tunably 'rugged'. Under the genetic operators of bit-flipping mutation or recombination, NK landscapes produce multiple domains of attraction for the evolutionary dynamics. NK landscapes have been used in models of epistatic gene interactions, coevolution, genome growth, and Wright's shifting balance model of adaptation. Theory for adaptive walks on NK landscapes has been derived, and generalizations that extend beyond Kauffman's original framework have been utilized.

B2.7.2.1 Introduction

A very short time after the first mathematical models of Darwinian evolution were developed, Sewall Wright (1932) recognized a deep property of *population genetic dynamics*: when fitness interactions exist between genes, the genetic composition of a population can evolve into multiple domains of attraction. The specific fitness interaction is *epistasis*, where the effect on fitness from altering one gene depends on the allelic state of other genes (Lush 1935). Epistasis makes it possible for the population to evolve toward different combinations of alleles, depending on its initial genetic composition. (Wright's framework also included the complication of diploid genetics, which augments the fitness interactions that produce multiple attractors.) A2.2

Wright thus found a conceptual link between a microscopic property of organisms—fitness interactions between genes—and a macroscopic property of evolutionary dynamics—multiple population attractors in the space of genotypes. To illustrate this situation, Wright invoked the metaphor of a landscape with multiple peaks, in which a population would evolve by moving uphill until it reached its local fitness peak. This metaphor of the 'adaptive landscape' is the general term used to describe multiple domains of attraction in evolutionary dynamics.

Wright was specifically interested in how populations could escape from local fitness peaks to higher ones through stochastic fluctuations in small population subdivisions. His was thus one of the earliest conceptions of a stochastic process for the optimization of multimodal functions.

Stuart Kauffman devised the *NK fitness landscape* model to explore the way that epistasis controls the ruggedness of an adaptive landscape (Kauffman and Levin 1987, Kauffman 1989). Kauffman wanted to specify a family of fitness functions whose ruggedness could be 'tuned' by a single parameter. He did this by building up landscapes from multiple 'atoms' of maximal epistasis.

The NK model is a stochastic method for generating fitness functions, $F : \{0, 1\}^N \mapsto \Re+$, on binary strings, $\boldsymbol{x} \in \{0, 1\}^N$, where the genotype $\boldsymbol{x}$ consists of N loci, with two possible alleles at each locus x_i. (As such, it is an example of a *random field* model elaborated upon by Stadler and Happel (1995).) It has two basic components: a structure for gene interactions, and a way this structure is used to generate a fitness function for all the possible genotypes.

The gene interaction structure is created as follows: the genotype's fitness is the average of N fitness components F_i contributed by each locus i. Each gene's fitness component F_i is determined by its own allele, x_i, and also the alleles at K other epistatic loci (so K must fall between zero and $N - 1$). Thus, the fitness function is:

$$F(\boldsymbol{x}) = \frac{1}{N}\sum_{i=1}^{N} F_i(x_i; x_{i_1}, \ldots, x_{i_K}) \tag{B2.7.7}$$

where $\{i_1, \ldots, i_K\} \subset \{1, \ldots, i-1, i+1, \ldots, N\}$. These K other loci could be chosen in any number of ways from the N loci in the genotype. Kauffman investigated two possibilities: *adjacent neighborhoods*,

† The author thanks the Maui High Performance Computing Center for generously hosting him as a visiting researcher.

where the K genes nearest to locus i on the chromosome are chosen; and *random neighborhoods*, where these K other loci are chosen randomly on the chromosome. In the adjacent neighborhood model, the chromosome is taken to have periodic boundaries, so that the neighborhood wraps around the other end when it is near the terminus.

Epistasis is implemented through a 'house of cards' model of fitness effects (Kingman 1978, 1980): whenever an allele is changed at one locus, all of the fitness components with which the locus interacts are changed, without any correlation to their previous values. Thus, a mutation in any one of the genes affecting a particular fitness component is like pulling a card out of a house of cards—it tumbles down and must be rebuilt from scratch, with no information passed on from the previous value.

Kauffman implemented this by generating, for each fitness component, a table of 2^{K+1} numbers for each possible allelic combination for the $K+1$ loci determining that fitness component. These numbers are independently sampled from a uniform distribution on $[0, 1)$. (See section B2.7.2.4 for alternative implementations of this scheme.)

The consequence of this independent resampling of fitness components is that the fitness function develops conflicting constraints: a mutation at one gene may improve its own fitness component but decrease the fitness component of another gene with which it interacts. Furthermore, if the allele at another interacting locus changes, an allele that had been optimal, given the alleles at the other loci, may no longer be optimal. Thus, epistatic interactions produce 'frustration' in trying to optimize all genes simultaneously, a concept borrowed from the field of spin glasses, of which NK landscapes are an example (Anderson 1985).

B2.7.2.2 Evolution on NK landscapes

The definition given by Kauffman for the NK landscape is simply a fitness function on a data structure. The genetic operators that manipulate these data structures in creating variants are not explicitly included in the NK landscape specification. However, nothing can be said about the evolutionary dynamics until the genetic operators are defined. A change in the genetic operator will effectively define a new adaptive landscape (Altenberg 1994a, 1995, Jones 1995a, b). The NK structure was defined with the 'natural' operators in mind: bit-flipping mutation, and recombination between strings. The magnitude of mutation and recombination rates also has a fundamental effect on the population dynamics.

One of the main differences between evolutionary algorithms and evolutionary genetics is relative time spent during transient, as opposed to near-equilibrium, phases of the dynamics. Biological populations have been running for a long time, and so their genetic compositions are relatively converged (Gillespie 1984); whereas in evolutionary algorithms, it is typical that initial populations are random over the search space, and so, for much of their dynamics, the populations are far from equilibrium.

The dynamics of nearly converged populations under low mutation rate can be approximated by *one-mutant adaptive walks* (Maynard Smith 1970, Gillespie 1984). The population is taken as fixed on a single genotype, and occasionally a fitter genotype is produced which then goes to fixation. The approximation assumes that the time it takes for the mutant to go to fixation is short compared to the time epochs between substitutions.

In implementing one-mutant adaptive walks, an initial genotype is chosen, and the fitnesses of all of the genotypes that can be produced by a single bit flip are sampled. A fitter variant (or the fittest, in the case of *greedy* or *myopic* walks) is selected, and the process is reiterated. When all of the one-mutant neighbors of a genotype are less fit than it, the walk terminates.

Results for one-mutant adaptive walks. The following is a synopsis of the results of Kauffman (1993), Weinberger (1991), and Fontana *et al* (1993) for one-mutant adaptive walks on NK landscapes.

For $K = 0$, the fitness function becomes the classical additive multilocus model.

(i) There is a single, globally attractive genotype.

(ii) The adaptive walk from any genotype in the space will proceed by reducing the Hamming distance to the optimum by one each step, and the number of fitter one-mutant neighbors is equal to this Hamming distance. Therefore, the expected number of steps to the global optimum is $N/2$.

(iii) The fitnesses of one-mutant neighbor genotypes are highly correlated, as $N-1$ of the N fitness components are unaltered between the neighbors.

For $K = N - 1$, the fitness function is equivalent to the random assignment of fitnesses over the genotype space.

(i) The probability that a genotype is a local optimum is $1/(N + 1)$.
(ii) The expected total number of local optima is $2^N/(N + 1)$.
(iii) The expected fraction of one-mutant neighbors that are fitter decreases by $1/2$ each step of the adaptive walk.
(iv) The expected length of adaptive walks is approximately $\ln(N - 1)$.
(v) The expected number of mutants tested before reaching a local optimum is $\sum_{i=0}^{\log_2(N-1)-1} 2^i$.
(vi) As N increases, the expected fitness of the local optimum reached from a random initial genotype decreases toward the mean fitness of the entire genotype space, 0.5. Kauffman (1993) calls this the *complexity catastrophe*.

For intermediate K, it is found that:

(i) For K small, the highest local optima share many of their alleles in common. As K increases, this allelic correlation among local optima falls away, and more rapidly for random neighborhoods than adjacent neighborhoods.
(ii) For K large, the fitnesses of local optima are distributed with an asymptotically normal distribution with mean approximately

$$\mu + \sigma \left(\frac{2 \ln(K + 1)}{K + 1} \right)^{1/2}$$

and variance approximately

$$\frac{(K + 1)\sigma^2}{N[K + 1 + 2(K + 2)\ln(K + 1)]}$$

where μ is the expected value of F_i, and σ^2 its variance. In the case of the uniform distribution, $\mu = 1/2$ and $\sigma = (1/12)^{1/2}$.
(iii) The average Hamming distance between local optima, which is roughly twice the length of a typical adaptive walk, is approximately

$$\frac{N \log_2(K + 1)}{2(K + 1)}.$$

(iv) The fitness correlation between genotypes that differ at d loci is

$$R(d) = \left(1 - \frac{d}{N}\right)\left(1 - \frac{K}{N - 1}\right)^d$$

for the random neighborhood model, and

$$R(d) = 1 - \frac{K + 1}{N} d + \frac{1}{\binom{N}{d}} \sum_{j=1}^{\min(K, N+1-d)} (K - j + 1) \binom{N - j - 1}{d - 2}$$

for the adjacent neighborhood model.

Results for full population dynamics. Most studies using NK models have investigated adaptive walks on the landscape. A notable exception is the study of Wright's shifting balance process using an NK landscape (Bergman *et al* 1995). In this study, the genotypes are distributed on a one-dimensional spatial array, and mating and dispersal along the array are studied with different length scales. Mutation rates of 10^{-4} per locus per reproduction, and single-point recombination rates of 0.01 or 0.1 per chromosome per reproduction are examined. The NK fitness function is extended to diploid genotypes.

This model produced rich interactions of dispersal distance, recombination rate, and K with the mean fitness that is attained during evolution. For highly rugged landscapes recombination made little difference in fitness attained, whereas at lower values of K, recombination could either improve or reduce the final fitness depending in a nonlinear way on the other parameters. The results support Wright's original theory: the greater the ruggedness of the landscape, the larger is the improvement in evolutionary optimization that population subdivision provides.

B2.7.2.3 Generalized NK maps

The epistatic interaction structure described by Kauffman can be seen to be special cases of more general interaction structures. Although Kauffman conceives of each gene as contributing a fitness component, inspection of equation (B2.7.7) shows, in fact, that a gene and the other K loci that interact with it are all symmetric in their effect on the fitness component. Therefore, one can remove the identification of one gene with one fitness component, and conceive of a set of N genes and a set of f fitness components and a map between them. This generalized fitness function is

$$F(\boldsymbol{x}) = \frac{1}{f}\sum_{i=1}^{f} F_i(x_{j_{1(i)}}, x_{j_{2(i)}} \ldots, x_{j_{p(i)}})$$

where $p(i)$ is the number of genes affecting fitness component i (its *polygeny*) and $\{j_{1(i)}, j_{2(i)} \ldots, j_{p(i)}\} \subset \{1, \ldots, N\}$. The index sets $\{j_{1(i)}, j_{2(i)} \ldots, j_{p(i)}\}$ comprise a gene–fitness map, that can be represented as a matrix,

$$\mathbf{M} = [m_{ij}] \qquad i = 1, \ldots, f \qquad j = 1, \ldots, N \tag{B2.7.8}$$

of indices $m_{ij} \in \{0, 1\}$, where $m_{ij} = 1$ indicates that gene j affects fitness component i. The rows of $\mathbf{M}$, $\boldsymbol{g}_i = [m_{ij}]$, $j = 1, \ldots, N$, give the genes controlling each fitness component i. The columns of $\mathbf{M}$, $\boldsymbol{p}_j = [m_{ij}]$, $i = 1, \ldots, f$, give the fitness components controlled by each gene j. These vectors, $\boldsymbol{p}_j$, represent each gene's *pleiotropy*. It is assumed that each gene affects at least one fitness component, and vice versa.

The fitness components F_i can be represented with a single uniform pseudorandom function U:

$$F_i(\boldsymbol{x}) = U(\boldsymbol{x} \circ \boldsymbol{g}_i, \boldsymbol{g}_i, i) \sim \text{uniform on } [0, 1) \tag{B2.7.9}$$

where $U : \{0, 1\}^N \times \{0, 1\}^N \times \{1, \ldots, N\} \mapsto [0, 1)$ and $\circ$ is the Hadamard product:

$$x \circ g_i = \begin{bmatrix} x_1 m_{i1} \\ x_2 m_{i2} \\ \vdots \\ x_N m_{iN} \end{bmatrix}.$$

A change in any of the three arguments i, $\boldsymbol{g}_i$, or $\boldsymbol{x} \circ \boldsymbol{g}_i$ gives a new value for $U(\boldsymbol{x} \circ \boldsymbol{g}_i, \boldsymbol{g}_i, i)$ that is uncorrelated with the old value. See section B2.7.2.4 for methods of implementing $U(x \circ g_i, g_i, i)$.

Some illustrations of this generalization of the NK mold are given in figure B2.7.2. The first two maps are standard Kauffman NK maps, which require the diagonal to be filled. The third is a map that produces a 'block model' of Perelson and Macken (1995). The fourth is an example of a map grown by selective gene addition, a process which produces highly nongeneric NK landscapes (see section B2.7.2.7; Altenberg 1994b).

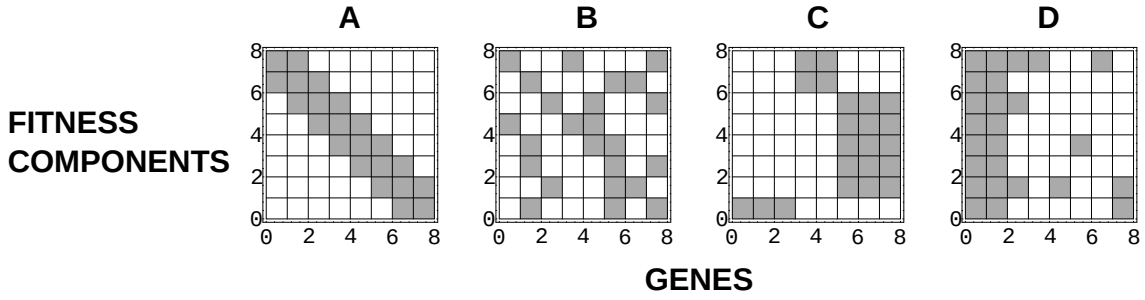


Figure B2.7.2. Four different gene–fitness interaction maps. Dark entries are where the gene affects the fitness component. (A) Kauffman's adjacent neighborhood, $N = 8, K = 2$; (B) Kauffman's random neighborhood, $N = 8, K = 2$; (C) a Perelson and Macken (1995) 'block' map; (D) a map evolved through genome growth (Altenberg 1994b).

The block model presents an opportunity to study recombination operators, which has not yet been utilized in the literature. Recombination between blocks is effectively operating on a smooth landscape,

whereas mutation will still experience frustration, to a degree that depends on the size of the block. One may conjecture that a relation could be elucidated between the 'blockiness' of the gene interaction map, and the relative effectiveness of recombination as a genetic operator in comparison to mutation. The blockiness of a generalized NK landscape could serve as another tunable parameter for investigating recombination as an evolutionary strategy.

B2.7.2.4 Implementation details

Kauffman's algorithm for generating an NK landscape requires storing the values for all of the $2^{(K+1)}$ possible allelic combinations of each fitness component. Since there are N fitness components, this approach requires storage of $2^{(K+1)}N$ numbers. For small K, this poses no problem. But with large K and N, storage and computation become formidable. With 32 genes and $K = 22$, a gigabyte of storage is needed (4 bytes/real $\times$ 32 $\times$ $2^{(22+1)}$). Yet, depending on the evolutionary algorithm used, often many of these numbers will never be called during the run. So one could instead create fitness component values as they are needed, and store them for later access (using, for example, a binary tree structure (Wirth 1975)).

A simple method (used by Altenberg (1994b)) which requires more computation but no storage, is to use a pseudorandom function to compute fitness components as they are called:

$$\Psi : \{0, 2^W - 1\} \mapsto \{0, 2^W - 1\}$$

where W is the bit width of the integer representation. Ψ can be used to implement equation (B2.7.9) as:

$$F_i(\boldsymbol{x}) = 2^{-W}\Psi\{(\boldsymbol{x} \circ \boldsymbol{g}) \wedge \Psi[\boldsymbol{g} \wedge \Psi(i \wedge t)]\}$$

where t is the integer seed of the run, $\wedge$ is the bitwise exclusive-or operator, and the bit strings $\boldsymbol{g}$ and $\boldsymbol{x}$ are represented as integers.

One must be careful in the choice of algorithms for Ψ. Park-Miller random number generators are unsuitable for Ψ, as there are correlations between input bits and output bits. However, the 'pseudo data-encryption-standard' algorithm, $ran4$ (Press *et al* 1992), works well as Ψ for genomes of length $L \leq 32$, and can be extended for larger genomes.

B2.7.2.5 Computational complexity of NK landscapes

The computational complexity of finding the optimum genotype in an NK landscape has been analyzed by Weinberger (1996) and Thompson and Wright (1996). The algorithms they use for the proofs depend only on the epistatic structure of the gene interaction map, and not the statistical assignment of fitnesses.

Weinberger provides a dynamic programming algorithm that finds the optimum genotype of an NK landscape with adjacent neighborhoods for any K. He is also able to reduce the NK optimization problem with random $K \geq 3$ neighborhoods to the well-known 3SAT problem (Garey and Johnson 1979). Thompson and Wright were able to reduce the NK optimization problem with random $K = 2$ neighborhoods to the 2SAT problem (Garey and Johnson 1979). These techniques prove the following theorems.

Theorem B2.7.1 (Weinberger). The NK optimization problem with adjacent neighborhoods is solvable in $\mathcal{O}(2^K N)$ steps, and is thus in $\mathcal{P}$.

Theorem B2.7.2 (Weinberger). The NK optimization problem with random neighborhoods is $\mathcal{NP}$ complete for $K \geq 3$.

Theorem B2.7.3 (Thompson and Wright). The NK optimization problem with random $K = 1$ neighborhoods is solvable in polynomial time.

Theorem B2.7.4 (Thompson and Wright). The NK optimization problem with random $K = 2$ neighborhoods is $\mathcal{NP}$ complete. Moreover, for a generalized $K = 1$ map with no requirement that $m_{ii} = 1$ for all i (in equation (B2.7.8)), the NK optimization problem is $\mathcal{NP}$ complete.

The Fourier expansion analysis of NK landscapes by Stadler and Happel (1995) corroborates the difference between random and adjacent neighborhood models: with adjacent neighborhoods, only the

first $K+1$ Fourier components contribute, while all contribute in the random neighborhood model. Thus, even though adaptive walks on NK landscapes do not show much difference between adjacent neighborhood and random neighborhood models, the computational complexity of these two families of landscapes is quite different.

B2.7.2.6 Application to coevolution

Kauffman (1993) used the NK model to frame a novel hypothesis about coevolving ecosystems: that they are poised on the 'edge of chaos', exhibiting a form of self-organized criticality (Bak *et al* 1988). In his model, Kauffman let the genes of other organisms interact with a gene's fitness component. Hence, evolution of one organism's gene alters the fitness landscape of other organisms. Kauffman used adaptive walks as the dynamics of the coevolving species. He found that smooth landscapes when coupled together produce chaotic dynamics—the 'red queen' hypothesis, that organisms have to evolve as fast as they can just to stay in the same place (Van Valen 1973), and the average fitness of organisms in the ecosystem is low. On the other extreme, in very rugged landscapes, the likelihood of the species reaching a local equilibrium is very high, but these equilibria are of low average fitness for the ecosystem. There is a threshold level of ruggedness that results in criticality of the dynamics, with a spectrum of 'avalanches' of coevolutionary change, the larger the avalanche, the less frequent. This critical value appears to give the highest average fitness over the ecosystem.

B2.7.2.7 Application to the representation problem

The generalized NK model has been applied to the representation problem in evolutionary computation: how to represent the objects in the search space so that genetic operators can have a reasonable chance of producing fitter variants when acting on the representation. One method proposed for producing good representations is to evolve the representation itself through a process of selective genome growth (Altenberg 1994b).

In the model, the gene-fitness map $\mathbf{M}$ is built up gene by gene: new genes with randomly chosen connections to the fitness components (i.e. new columns of $\mathbf{M}$ with randomly chosen entries $\in \{0, 1\}$ were added to the genome only if they produced a fitness increase. It was found that as more genes were added and the fitness increased, selection for genes with low pleiotropy (affecting few functions) became more intense. An example of an evolved NK map is shown in figure B2.7.2(D). The fitness peaks of the resulting NK maps were several standard deviations above the fitness distribution for generic NK landscapes with the same interaction maps.

The NK model is thus used as an abstraction for the way representations produce epistatic interactions between genes. It was suggested that the method of selective genome growth which was able to produce highly evolvable NK landscapes might be applicable toward more general representation problems.

B2.7.3 Correlation analysis

Bernard Manderick

Abstract

Correlation analysis is a set of techniques that are intended to characterize the difficulty of a search problem for a genetic algorithm (or any other search technique) by exploiting the fitnesses between neighboring search points and the correlation of the fitnesses between parents and their offspring. Three measures and their practical uses are discussed based on correlation analysis: the autocorrelation function of a fitness landscape, the fitness correlation coefficients of genetic operators and the fitness–distance correlation.

B2.7.3.1 Fitness landscapes

A fitness landscape, $\mathrm{FL} = ((S, d), f)$, is the combination of a metric space (S, d) and a fitness function f defined over that space. We assume here that f is defined for all $s \in S$ and that f takes only nonnegative real values; that is, $f(s) \geq 0$. A metric space (S, d) is a set S provided with a metric or distance d. A metric d is a real-valued map defined on $S \times S$ which fulfills the following conditions for all $s_1, s_2 \in S$:

(i) $d(s_1, s_2) \geq 0$ and $d(s_1, s_2) = 0$ if and only if $s_1 = s_2$;

(ii) $d(s_1, s_2) = d(s_2, s_1)$, i.e. d is symmetric; and

(iii) $d(s_1, s_3) \leq d(s_1, s_2) + d(s_2, s_3)$, i.e. d satisfies the triangle inequality.

Many search problems define a corresponding fitness landscape. We will illustrate this with two examples: fitness landscapes defined over hypercubes and fitness landscapes defined by combinatorial optimization problems such as the traveling salesman problem or job shop scheduling problems.

In many genetic algorithm (GA) applications we have a fitness function f which associates a fitness $f(b)$ with each bit string $b = b_{N-1}b_{N-2}\ldots b_2b_1b_0$ of length N. Moreover, we can define a distance on the set B of bit strings, for example, the Hamming distance d_H between two bit strings $b, b' \in B$ which is defined as the number of bit positions in which b and b' differ. For instance, the Hamming distance between the two 5-bit strings 01001 and 11011 is two since these bit strings differ in the first and fourth positions. Once we have provided the set B with the distance d_H, the resulting metric space (B, d_H) is called the hypercube of dimension N since each bit string b has N neighbors b' at a Hamming distance one. For instance, the neighborhood of the string 00000 consists of the five strings 10000, 01000, 00100, 00010, and 00001. Well-known examples of fitness landscapes defined over N-dimensional hypercubes (B, d_H) are the NK landscapes (Kauffman 1993) discussed in section B2.7.2.

The *traveling salesman problem* (TSP), like many other combinatorial problems, defines a fitness landscape in a similar way. The TSP is defined as follows. Given n cities $c_1, c_2, \ldots, c_n$ and their mutual distances $l(c_i, c_j)$, $i, j = 1, \ldots, n$, find a tour which visits all cities just once and brings the traveling salesman back to his starting point in such a way that the total distance of the tour, also called its cost, is minimized. G9.5

The search space Π consists of all possible permutations of the n cities $c_i, i = 1, \ldots, n$, since a permutation tells us in what order the cities have to be visited. The cost c of a solution $\pi = (c_{i_1} \ldots c_{i_{j-1}} c_{i_j} c_{i_j} \ldots c_{i_n})$ is the sum of the distances between every pair of adjacent cities; that is, $c(\pi) = \sum_{j=1}^{n-1} l(c_{i_j}, c_{i_{j+1}}) + l(c_{i_n}, c_{i_1})$. This cost function c has to be minimized, and it can easily be transformed to a fitness function which then has to be maximized: $f = 1/c$.

We can define a distance d_{inv} on the space of all permutations Π using the inversion operation. Take an arbitrary permutation $\pi = (c_{i_1} \ldots c_{i_{l-1}} c_{i_l} c_{i_{l+1}} \ldots c_{i_{k-1}} c_{i_k} c_{i_{k+1}} \ldots c_{i_n})$; the result of the inversion starting at c_{i_l} and ending at c_{i_k} is obtained by inverting the subsequence of π between and including these two: $\pi' = (c_{i_1} \ldots c_{i_{l-1}} c_{i_k} c_{i_{k-1}} \ldots c_{i_{l+1}} c_{i_l} c_{i_{k+1}} \ldots c_{i_n})$. Since we can choose the starting and ending points of the inversion freely, there are $n(n-1)$ possible inversions for each permutation π which are called the neighbors of π. The distance $d_{inv}(\pi_1, \pi_2)$ between any two permutations is defined as the minimal number of inversions needed to transform π_1 into π_2. It is easy to verify that d_{inv} defines a metric on the search space Π. Finally, since we can associate with each permutation π its fitness $f(\pi)$, the TSP also defines a fitness landscape. Part of a fitness landscape corresponding to a nine-city problem is shown in figure B2.7.3.

The long-term goal of correlation analysis is to find out what landscape features make the search easy or difficult for the GA and how the GA uses information obtained from that landscape to guide its search.

Several measures based on fitness correlations have been defined and their relation to problem difficulty and GA performance has been studied. In the next sections, we discuss the autocorrelation function, the correlation length, the correlation coefficient of genetic operators, and the fitness–distance correlation.

B2.7.3.2 The autocorrelation function

A first idea to analyze GA performance consists of calculating the autocorrelation function of a random walk in the landscape. Given a fitness landscape $((S, d), f)$, where (S, d) is a metric space and f is a fitness function, select a random start point s_0 and select a random neighbor s_1, i.e. $d(s_0, s_1) = 1$, repeat this process N times, and collect the fitnesses $f(s_i)$ of the encountered search points $s_i, i = 0, \ldots, N$. This way, a series $F = (f(s_0), f(s_1), \ldots, f(s_{i-1}), f(s_i), f(s_{i+1}), \ldots, f(s_N))$ is obtained in which the pairs s_{i-1}, s_i and s_i, s_{i+1}, for $i = 1, \ldots, N-1$, are neighboring search points.

The autocorrelation function ρ of the random walk is defined as

$$\rho(h) = \frac{1}{s_F^2} R(h) \tag{B2.7.10}$$

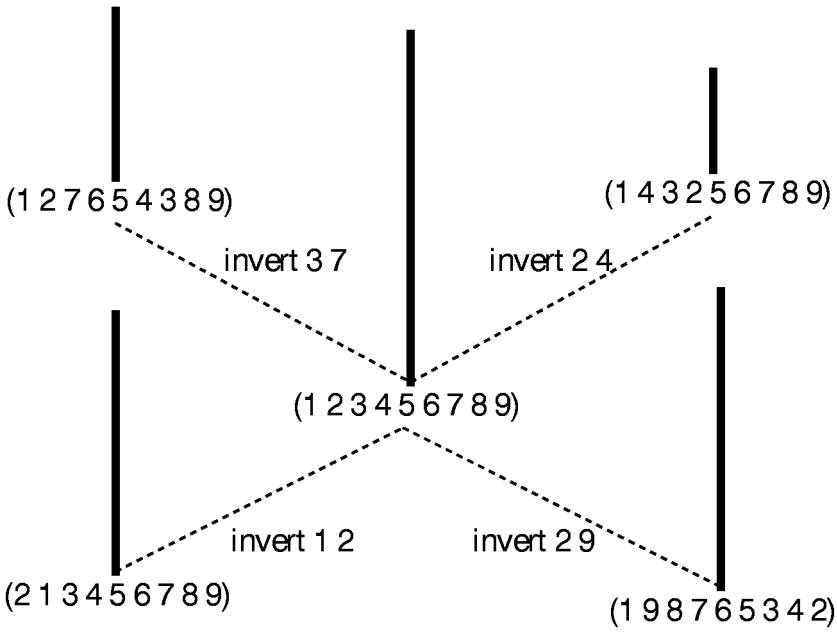


Figure B2.7.3. Part of a fitness landscape defined by a nine-city TSP when the inversion distance d_{inv} is used. Dotted lines represent neighborhood relations; labels of these lines give the inversion which transforms neighboring points into each other. Solid lines represent the fitnesses of the corresponding search points; lengths are proportional to the fitnesses.

where $s_F^2 = (1/N)\sum_{i=0}^{N}(f(s_i) - m_F)^2$ is the variance and $R(h)$ is the autocovariance function of the series F. For all h, $R(h)$ can be estimated by

$$R(h) = \frac{1}{N}\sum_{i=0}^{N-h}(f(s_i) - m_F)(f(s_{i+h}) - m_F) \tag{B2.7.11}$$

$$m_F = \frac{1}{N+1}\sum_{i=0}^{N} f(s_i) \tag{B2.7.12}$$

where $N > 0$ and $0 \leq h < N$. It can be shown that $-1 \leq \rho(h) \leq 1$ and $\rho(0) = 1$. The autocorrelation function $\rho(h)$ expresses for each distance h how correlated search points are which are at a distance h from each other.

It can be shown that the autocorrelation function for many optimization problems is an exponentially decreasing function; that is, $\rho(h) = \mathrm{e}^{-ah}$ where a is a constant. For example, this is the case for the NK landscapes (Weinberger 1990), for TSPs where the cities are randomly distributed over a square (Stadler and Schnabl 1992), and for random graph bipartition problems (Stadler and Happel 1992).

In this case, one can define the correlation length τ as the distance h where $\rho(h) = 1/2$. The larger the correlation length τ the more correlated and smoother the fitness landscape is. Small τ correspond to rugged fitness landscapes. It has been shown empirically (Manderick *et al* 1991) that on the NK landscapes there is a strong relation between the correlation length τ and the GA performance on these landscapes: the smaller τ, the harder the corresponding landscape is for the GA. The autocorrelation function ρ and the correlation length τ therefore provide a rough indication of how difficult a landscape is.

B2.7.3.3 The fitness correlation coefficient

A second way to analyze GA performance consists of calculating the fitness correlation coefficient of a genetic operator. Suppose we have an g-ary genetic operator OP. This means that OP takes g parents $p_1, p_2, \ldots, p_g$ and produces one offspring c: for example, *mutation* is an unary operator since it takes one parent to produce one offspring while *crossover* is a binary operator since it usually takes two parents to produce one offspring. The correlation coefficient ρ_{OP} of a genetic operator OP measures the fitness correlation between parents and the offspring that they produce.

C3.2 C3.3.1

Formally, if the operator OP is g-ary then take N sets, $i = 1, \ldots, N$, of g parents $\{p_{i_1}, p_{i_2}, \ldots, p_{i_g}\}$ with fitnesses $\{(f(p_{i_1}), f(p_{i_2}), \ldots, f(p_{i_g})\}$ and mean fitness $m_{f_{p_i}} = (1/g)\sum_{j=1}^{g} f(p_{i_j})$, and generate the corresponding offspring c_i with fitness $f(c_i)$ using OP. The correlation coefficient ρ_{OP} measures the correlation between the series $F_p = (m_{f_{p_1}}, \ldots, m_{f_{p_N}})$ and $F_c = (f(c_1), \ldots, f(c_N))$ and is calculated as follows:

$$\rho_{\mathrm{OP}} = \frac{c_{F_p F_c}}{s_{F_p} s_{F_c}} \tag{B2.7.13}$$

where $c_{F_p F_c}$ is the covariance between the series F_p and F_c, and s_{F_p} and s_{F_c} are the standard deviations of F_p and F_c. The correlation coefficient ρ_{OP} measures how correlated the landscape appears for the corresponding operator OP.

One might expect that the larger the coefficient ρ_{OP} of an operator the more correlated a landscape for this operator and the more useful it will be in genetic search. This hypothesis has been confirmed on two combinatorial optimization problems, the TSP and *job flow scheduling* problems (Manderick *et al* 1991). Moreover, calculating the ρ_{OP} for each operator provides an easy way to find the combination of mutation and crossover operators which gives the best GA performance. We illustrate this on a standard TSP problem (see Oliver *et al* 1987). In table B2.7.1 three mutation operators are shown together with their correlation coefficient for this TSP. Manderick *et al* (1991) have shown that the correlation coefficient ranks the mutation operators according to their usefulness for the GA. So, for the TSP the Reverse mutation is the best operator. Similar results exist for the crossover operators. So, the correlation coefficient provides an easy way to select the operators for a given problem without trying all possible combinations for the problem at hand.

F1.5

B2.7.3.4 Fitness–distance correlation

A last way to analyze GA performance consists of calculating the fitness–distance correlation (FDC) (Jones and Forrest 1995). In order to calculate this measure the global optimum of the optimization problem has to be known. The FDC measures the correlation between the fitnesses of search points and the distances of these points to the (nearest) global optimum.

Suppose we have N search points $\{s_1, s_2, \ldots, s_N\}$ sampled at random together with their distances $\{d_1, d_2, \ldots, d_N\}$ to the global optimum, then the FDC coefficient ρ_{FDC} is defined as

$$\rho_{\mathrm{FDC}} = \frac{c_{FD}}{s_F s_D} \tag{B2.7.14}$$

where

$$c_{FD} = \frac{1}{N}\sum_{i=1}^{N}(f(s_i) - m_F)(d_i - m_D) \tag{B2.7.15}$$

is the covariance of the series $F = (f(s_1), \ldots, f(s_N))$ and $D = (d_1, \ldots, d_N)$, and s_F, s_D, m_F, and m_D are the standard deviations and the means of F and D, respectively.

It can be shown that $-1 \leq \rho_{\mathrm{FDC}} \leq 1$. Note that maximal correlation corresponds to $\rho_{\mathrm{FDC}} = -1$ since then search points at short distances are highly correlated in fitness. Using the FDC coefficient ρ_{FDC}, three classes of problem difficulty can be defined:

- easy: $\rho_{\mathrm{FDC}} \leq -0.15$
- difficult: $-0.15 < \rho_{\mathrm{FDC}} < 0.15$
- misleading: $\rho_{\mathrm{FDC}} \geq 0.15$.

Table B2.7.1. The Swap, Reverse and Remove-and-Reinsert mutations of the tour (1 2 3 4 5 6 7 8 9 10) when the fourth and eighth cities are selected. The cities in the mutant tour that differ from the parent one are shown in bold.

	Swap	Reverse	Remove-and-Reinsert
Mutant	(1 2 3 **8** 5 6 7 **4** 9 10)	(1 2 3 **8 7 6 5 4** 9 10)	(1 2 3 **5 6 7 8 4** 9 10)
ρ_{operator}	0.77	0.86	0.80

So far, FDC has only been applied to fitness landscapes defined on hypercubes. Jones and Forrest (1995) show that when a problem is easy, difficult, or misleading according to its ρ_{FDC} then this is also the case for the GA. The FDC coefficient ρ_{FDC} is thus able to correctly classify problem difficulty for the GA. Moreover, it could 'explain' unexpected results with the royal road functions and correctly classify some deceptive problems as quite easy while according to the *schema theorem* these problems should be hard. B2.5.2

B2.7.4 Test landscapes

Thomas Bäck and Zbigniew Michalewicz

Abstract

The availability of appropriate, standardized sets of test functions is of high importance for assessing evolutionary algorithms with respect to their effectiveness and efficiency. This section summarizes the properties of test functions and test suites which are desirable to investigate the behavior of evolutionary algorithms for continuous parameter optimization problems. A test suite should contain some simple unimodal function and multimodal functions with a large number of local optima, which are high dimensional and scalable, and incorporate constraints in some cases. A regular arrangement of local optima, separability of the objective function, decreasing difficulty of the problem with increasing dimensionality, and a potential bias introduced by locating the global optimum at the origin of the coordinate system are identified as properties of multimodal objective functions which are neither representative of arbitrary problems nor well suited for assessing the global optimization qualities of evolutionary algorithms. The section concludes by a presentation and discussion of some of the most prominent test functions used by the evolutionary computation community.

B2.7.4.1 Properties of test functions

Just as for any optimization algorithms, evolutionary algorithms need to be assessed concerning their efficiency and effectiveness for optimization purposes. Following Schwefel (1995), we use the term *efficiency* in the sense of convergence velocity (speed of approach to the objective), while *effectiveness* characterizes the reliability of the algorithm working under varying conditions (sometimes, the term *robustness* is also used). To facilitate a reasonably fair comparison of optimization algorithms in general and evolutionary algorithms in particular, a number of artificial test functions are typically used for an experimental comparison. Surprisingly, not only the experimental tests of *evolutionary programming* and *evolution strategies*, but also those of *genetic algorithms* are often performed on a set of continuous parameter optimization problems with functions of the form $f : M \subseteq \mathbb{R}^n \rightarrow \mathbb{R}$. Consequently, it is possible to identify some of the most widely used continuous parameter optimization problems, but almost no standard set of typical pseudo-Boolean objective functions $f : \mathbb{B}^\ell \rightarrow \mathbb{R}$ can be encountered in the literature. The *NK landscapes* and *royal road functions* are notable exceptions which have recently received some attention, but there is still a lack of a standardized set of test functions especially in the pseudo-Boolean case. Recently, Jones (1995a) presented a study that involved the comparison of 22 pseudo-Boolean functions based on measuring the correlation of fitness function values with distance to a global optimum. His study covers the complete range of functions, including NK landscapes, royal road functions, functions of unitation, and deceptive functions, and provides the most complete collection of pseudo-Boolean test cases used by researchers in the field of evolutionary algorithms. B1.4, B1.3, B1.2, B2.7.2, B2.7.5

Some prominent test suites of parameter optimization problems are those of De Jong (1975) and Schwefel (1977, 1995). De Jong presented five functions, which are all still used by the genetic algorithm community, while Schwefel's problem catalogue contains 68 objective functions covering a wide range of different topological characteristics.

While these test suites serve well as a repository for experimental comparisons, such comparisons are often restricted to a selection of a few out of the available functions. If such a selection is made, however, one should have in mind that it is important to cover various topological characteristics of landscapes in order to test the algorithms concerning efficiency *and* effectiveness. The following list summarizes some

of the properties of test suites and test functions that are reasonable in order to investigate the behavior of evolutionary algorithms.

(i) The test suite should include a few unimodal functions for comparisons of convergence velocity (efficiency).

(ii) The test suite should include several multimodal functions with a *large* number of local optima (e.g. a number growing exponentially with n, the search space dimension). These functions are intended to be representatives of the characteristics which are typical for real-world problems, where the best out of a number of optima is sought. When choosing multimodal functions for inclusion in a test suite, one should be careful because of the following facts:

   (a) Some test functions exhibit an extremely regular arrangement of local optima, which might favor particular operators of the evolutionary algorithm that exploit the regularity.

   (b) Some test functions obtained from a superposition of a 'global shape' (e.g. a quadratic bowl) and a finer structure of local optima might become *easier* for an algorithm to optimize when n is increased, which is counterintuitive because the dimension normally serves as the complexity parameter. As pointed out by Whitley *et al* (1995), the test function of Griewank (from Törn and Žilinskas 1989, p 186: $f(\boldsymbol{x}) = \sum_{i=1}^{n} x_i^2/d - \prod_{i=1}^{n} \cos(x_i/\sqrt{i}) + 1$, with $d = 4000$ and $-600 \leq x_i \leq 600$) has this property, because the local optima decrease in number and complexity as the dimension n is increased, and already for $n = 10$ the quadratic bowl dominates completely (notice that the parameter $d = 4000$ is specialized to dimension $n = 10$ and must grow exponentially for larger values of n).

   (c) The still prevalent choice to locate the global optimum at the origin of the coordinate system might implicitly bias the search in case of binary encoded object variables (Davis 1991). The bias might become even stronger when, in the case of canonical genetic algorithms, the interval $[u_i, v_i]$ of real values for the object variable x_i is symmetric around zero, that is, $u_i = -v_i$. Also by means of *intermediary recombination*, a bias towards the origin of the coordinate system is introduced when all variables are initialized in the interval $[-v_i, v_i]$ (Fogel and Beyer 1995). To circumvent this problem in the case of a global optimum at the origin, it is useful to check the evolutionary algorithm also for the problem $g(\boldsymbol{x}) = f(\boldsymbol{x} - \boldsymbol{a})$ for some $\boldsymbol{a} \in \mathbb{R}^n$ with $\boldsymbol{a} \neq \boldsymbol{0}$. C3.3.2

   (d) Multimodal test functions which are *separable*, that is, composed of a sum of one-dimensional subfunctions

$$f(\boldsymbol{x}) = \sum_{i=1}^{n} f_i(x_i) \tag{B2.7.16}$$

   are well suited for optimization by so-called *coordinate strategies* (e.g. see Schwefel 1995, pp 41–4), which change only one variable at each step. The test suite might contain such functions, but one should be aware of the fact that they are neither representatives of real-world problems nor difficult to optimize. Whitley *et al* (1995) go one step further and propose not to use separable functions at all. Again, the particular structure of f might favor certain operators of evolutionary algorithms, such as a mutation operator used in some variants of genetic algorithms which changes only the binary representation of a single, randomly selected object variable x_i at a time. It is known that an evolutionary algorithm using such an operator can optimize separable functions with $O(n \ln n)$ function evaluations (Mühlenbein and Schlierkamp-Voosen 1993), but line search achieves this in $O(n)$ function evaluations simply by omitting the random choice of the dimension $k \in \{1, \ldots, n\}$ to be mutated next.

   Provided that a rotated and scaled version of the problem is tested as well, separable functions may be part of a testbed for evolutionary algorithms. For a separable function $f(\boldsymbol{x})$, an $n \times n$ orthogonal matrix $\mathbf{T}$, and a diagonal matrix $\mathbf{S} = \mathrm{diag}(s_1, \ldots, s_n)$ with pairwise different $s_i > 0$, the problem $g(\boldsymbol{x}) = f(\mathbf{TS}\boldsymbol{x})$ is not separable, provided that $\mathbf{T}$ is not the unit matrix. Moreover, it is also possible to control the degree of non-separability: For example, if $\mathbf{T}$ is a bandmatrix of width three, then x_i is correlated with x_{i-1} and x_{i+1} for all $i \in \{2, \ldots, n-1\}$. A larger width introduces more correlation and thereby more nonseparability.

   Therefore, one should be aware of the possible bias introduced when using test functions having one or more of the properties listed above.

(iii) A test function with randomly perturbed objective function values models a typical characteristic of numerous real-world applications and helps to investigate the robustness of algorithms with respect

to noise. Ordinarily, the perturbation follows a normal distribution with an expectation of zero and a scalable variance of σ_δ^2.

Other types of noise might be of interest as well, such as additive noise according to a Cauchy distribution (such that the central limit theorem is not valid) or even multiplicative noise.

(iv) Real-world problems are typically constrained, such that the incorporation of *constraint handling techniques* into evolutionary algorithms is a topic of active research. Therefore, a test suite should also contain constrained problems with inequality constraints $g_j(\boldsymbol{x}) \geq 0$ and/or equality constraints $h_k(\boldsymbol{x}) = 0$ (notice that canonical genetic algorithms *require* the existence of inequality constraints $u_i \leq x_i \leq v_i$ for all object variables x_i for the purpose of encoding the object variables as a binary string—because these constraints are inherent to canonical genetic algorithms, they are called domains of variables in the remainder of this section). When experimenting with constrained objective functions, a number of additional criteria are also worth considering: C5

   (a) The number and type of constraints (e.g. linear or nonlinear ones) should vary for the test suite members.
   (b) Some constraints should be active at the optimum. This is an important criterion to determine whether or not a constraint-handling technique is able to locate an optimal solution even if it is located at the boundary between feasible and infeasible regions.
   (c) The test suite should contain constrained functions with various ratios between the sizes of the feasible search space and the whole search space. Typically, an estimate for this ratio can be obtained by a random sampling technique (see Chapter C5 for details). Obviously, a function with a small feasible region is more difficult to handle than a function where almost all points of the search space are feasible.

(v) The test suite should contain high-dimensional objective functions, because these are more representative of real-world applications. Furthermore, most low-dimensional functions (e.g. with $n = 2$) are not suitable as representatives of application problems where an evolutionary algorithm would be applied, because they can be solved to optimality with traditional methods. Most useful are test functions which are *scalable* with respect to n, i.e., which can be used for arbitrary dimensions.

These five basic properties correspond well with the requirements recently formulated by Whitley *et al* (1995), who proposed that test suites should contain nonlinear, nonseparable problems resistant to hillclimbing methods, they should contain scalable functions, and the test problems should have a canonical form. The canonical form requirement focuses on representational issues raised by using genetic algorithms for continuous parameter optimization purposes, where the coding scheme has to be specified exactly by giving the number of bits used to encode a single object variable, the type of decoding function (Gray code, standard binary-coded decimals), and the interval boundaries u_i, v_i ($\forall i \in \{1, \ldots, n\}$). A standardization of these parameters is of course mandatory for experimental comparisons, but it has nothing to do with the test problems themselves.

Whitley *et al* (1995) propose to build better test functions than those existing by constructing high-dimensional functions from lower-dimensional ones using *expansion* and *composition*. Starting with a nonlinear, two-dimensional $f(x_1, x_2)$, expansion yields $\tilde{f}(x_1, x_2, x_3) = f(x_1, x_2) + f(x_2, x_3) + f(x_3, x_1)$, while function composition with a separable $f(x_1, x_2, x_3) = g(x_1) + g(x_2) + g(x_3)$ and a function $h : \mathbb{R}^2 \to \mathbb{R}$ would yield $\tilde{f}(x_1, x_2, x_3) = g(h(x_1, x_2)) + g(h(x_2, x_3)) + g(h(x_3, x_1))$. Though these are certainly interesting techniques for the construction of *new* test functions, our focus is on the presentation and critical discussion of a few test problems out of those that have already been used to evaluate evolutionary algorithms.

In the following sections, this choice of test functions is presented and described in some detail.

B2.7.4.2 Unimodal functions

Sphere model (De Jong 1975, Schwefel 1977).

$$f(\boldsymbol{x}) = \sum_{i=1}^{n} x_i^2. \tag{B2.7.17}$$

Minimum: $f(\mathbf{0}) = 0$. Domains of variables for genetic algorithms: $-5.12 \leq x_i \leq 5.12$. For convergence velocity evaluation, this is the most widely used objective function.

Scaled sphere model (Schwefel 1988).

$$f(\boldsymbol{x}) = \sum_{i=1}^{n} i x_i^2. \tag{B2.7.18}$$

Minimum: $f(\mathbf{0}) = 0$. This function has been used by Schwefel to demonstrate the self-adaptation principle of strategy parameters in evolution strategies in case of n differently scaled axes.

Double sum (Schwefel 1977).

$$f(\boldsymbol{x}) = \sum_{i=1}^{n} \left(\sum_{j=1}^{i} x_j \right)^2. \tag{B2.7.19}$$

Minimum: $f(\mathbf{0}) = 0$. This function was introduced by Schwefel to demonstrate the self-adaptation of strategy parameters in evolution strategies, when variances and covariances of the n-dimensional normal distribution are learned.

Rosenbrock function (Rosenbrock 1960, De Jong 1975).

$$f(x_1, x_2) = 100(x_1^2 - x_2)^2 + (1 - x_1)^2. \tag{B2.7.20}$$

Minimum: $f(\mathbf{1}) = 0$. Domains of variables for genetic algorithms: $-5.12 \leq x_i \leq 5.12$. This function is two-dimensional; that is, it does not satisfy the condition of scalability (see point (v) in the earlier discussion on test suites and test functions). One might propose a generalized version $f(\boldsymbol{x}) = \sum_{i=1}^{n-1}(100(x_i^2 - x_{i+1})^2 + (1 - x_i)^2)$ (Hoffmeister and Bäck 1991), but it is not clear whether the topological characteristics of the optimum location at the bottom of a long, bent 'valley' remains unchanged.

B2.7.4.3 Multimodal functions

Step function (De Jong 1975).

$$f(\boldsymbol{x}) = \sum_{i=1}^{n} \lfloor x_i \rfloor. \tag{B2.7.21}$$

Domains of variables for genetic algorithms: $-5.12 \leq x_i \leq 5.12$. Minimum under these constraints: $x_i^* \in [-5.12, -5)$, $f(\boldsymbol{x}^*) = -6n$. The step function introduces plateaus to the topology, which make the search harder because slight variations of the x_i do not cause any change in objective function value. Bäck (1996) proposed to use a step function version of the sphere model, i.e., $f(\boldsymbol{x}) = \sum_{i=1}^{n} \lfloor x_i + 0.5 \rfloor^2$, where $x_i^* \in [-0.5, 0.5)$, $f(\boldsymbol{x}^*) = 0$, because this function does not require the existence of finite domains of variables to have an optimum different from minus infinity. As another alternative, one might use the absolute values $|x_i|$ in equation (B2.7.21).

Shekel's foxholes (De Jong 1975).

$$\frac{1}{f(\boldsymbol{x})} = \frac{1}{K} + \sum_{j=1}^{25} \frac{1}{c_j + \sum_{i=1}^{2}(x_i - a_{ij})^6}. \tag{B2.7.22}$$

The constants are defined according to

$$(a_{ij}) = \begin{pmatrix} -32 & -16 & 0 & 16 & 32 & -32 & \cdots & 0 & 16 & 32 \\ -32 & -32 & -32 & -32 & -32 & -16 & \cdots & 32 & 32 & 32 \end{pmatrix}$$

$K = 500$, and $c_j = j$. Minimum: $f(-32, -32) \approx 1$. Domains of variables for genetic algorithms: $-65.536 \leq x_i \leq 65.536$. Although often used in the genetic algorithm community, this function has some serious disadvantages: it is just two-dimensional (v), and the landscape consists of a flat plateau with 25 steep and narrow minima, arranged on a regular grid (ii.a) at the positions defined by the matrix $\mathbf{A}$, with $f(a_{1j}, a_{2j}) \approx c_j = j$.

Generalized Rastrigin function (Bäck and Hoffmeister 1991, Törn and Žilinskas 1989).

$$f(\boldsymbol{x}) = nA + \sum_{i=1}^{n} x_i^2 - A\cos(\omega x_i). \tag{B2.7.23}$$

The constants are given by $A = 10$ and $\omega = 2\pi$. Minimum: $f(\mathbf{0}) = 0$. Domains of variables for genetic algorithms: $-5.12 \leq x_i \leq 5.12$. This function was presented by Bäck and Hoffmeister (1991) as a generalization of Rastrigin's original definition, reprinted in Törn and Žilinskas (1989). It is not a particularly good test problem because the function is separable (ii.d) and the local minima are arranged on a regular grid (ii.a).

Generalized Ackley function (Bäck et al 1993, Ackley 1987).

$$f(\boldsymbol{x}) = -a\exp\left[-b\left(\frac{1}{n}\sum_{i=1}^{n} x_i^2\right)^{1/2}\right] - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(cx_i)\right) + a + \exp(1)\exp(1). \tag{B2.7.24}$$

Constants: $a = 20$, $b = 0.2$, $c = 2\pi$. Minimum: $f(\mathbf{0}) = 0$. Domains of variables for genetic algorithms: $-32.768 \leq x_i \leq 32.768$. Originally defined by Ackley as a two-dimensional test function, the general extension was proposed for example by Bäck *et al* (1993). The function is not separable, and there are no other disadvantages either except a regular arrangement of the minima (ii.a).

Fletcher and Powell function (Fletcher and Powell 1963).

$$f(\boldsymbol{x}) = \sum_{i=1}^{n}(A_i - B_i)^2 \tag{B2.7.25}$$

$$A_i = \sum_{j=1}^{n}(a_{ij}\sin\alpha_j + b_{ij}\cos\alpha_j) \tag{B2.7.26}$$

$$B_i = \sum_{j=1}^{n}(a_{ij}\sin x_j + b_{ij}\cos x_j). \tag{B2.7.27}$$

The constants $a_{ij}, b_{ij} \in [-100, 100]$ as well as $\alpha_j \in [-\pi, \pi]$ are randomly chosen and specify the position of the local minima (there are 2^n local minima in the range $-\pi \leq x_i \leq \pi$). Minimum: $f(\alpha) = 0$. Domains of variables for genetic algorithms: $-\pi \leq x_i \leq \pi$. This function has the advantage that local minima are randomly arranged and the function is scalable; for $n \leq 30$, the matrices **A** and **B** are tabulated by Bäck (1996). Alternatively, one could also define a simple pseudo-random number generator together with a certain seed to use.

B2.7.4.4 Constrained problems

Here, we present just four representative problems for the cases of having linear as well as nonlinear inequality constraints. Notice that the constrained problems are generally *not* scalable with respect to their dimensionality, because this would also require scalable constraints (which is certainly not an unsolvable problem, but was not considered in the known test problems). For further sources of constrained test problems, the reader is referred to the work of Hock and Schittkowski (1981), Michalewicz *et al* (1994), Michalewicz (1995), and Floudas and Pardalos (1990).

Problem B2.7.1 (Colville 1968).

$$\begin{aligned} f(\boldsymbol{x}) = {} & 100(x_2 - x_1^2)^2 + (1 - x_1)^2 + 90(x_4 - x_3^2)^2 + (1 - x_3)^2 \\ & + 10.1((x_2 - 1)^2 + (x_4 - 1)^2) + 19.8(x_2 - 1)(x_4 - 1) \end{aligned} \tag{B2.7.28}$$

subject to

$$-10.0 \leq x_i \leq 10.0 \qquad i = 1, 2, 3, 4.$$

Minimum: $f(1, 1, 1, 1) = 0$.

Problem B2.7.2 (Hock and Schittkowski 1981, Michalewicz 1995).

$$f(\boldsymbol{x}) = x_1 + x_2 + x_3 \tag{B2.7.29}$$

subject to

$$\begin{aligned}
&1 - 0.0025(x_4 + x_6) \geq 0 \\
&1 - 0.0025(x_5 + x_7 - x_4) \geq 0 \\
&1 - 0.01(x_8 - x_5) \geq 0 \\
&x_1 x_6 - 833.332\,52 x_4 - 100 x_1 + 83\,333.333 \geq 0 \\
&x_2 x_7 - 1250 x_5 - x_2 x_4 + 1250 x_4 \geq 0 \\
&x_3 x_8 - 1\,250\,000 - x_3 x_5 + 2500 x_5 \geq 0 \\
&100 \leq x_1 \leq 10\,000 \\
&1000 \leq x_i \leq 10\,000 \qquad i = 2, 3 \\
&10 \leq x_i \leq 1000 \qquad i = 4, \ldots, 8.
\end{aligned}$$

Minimum: $f(579.3167, 1359.943, 5110.071, 182.0174, 295.5985, 217.9799, 286.4162, 395.5979) = 7049.330\,923$. The problem has three linear and three nonlinear constraints; all six constraints are active at the global optimum. The ratio between the size of the feasible search space and the whole search space is approximately 0.001%.

Problem B2.7.3 (Hock and Schittkowski 1981, Michalewicz 1995).

$$\begin{aligned}
f(\boldsymbol{x}) = {} & x_1^2 + x_2^2 + x_1 x_2 - 14 x_1 - 16 x_2 + (x_3 - 10)^2 + 4(x_4 - 5)^2 \\
& + (x_5 - 3)^2 + 2(x_6 - 1)^2 + 5 x_7^2 + 7(x_8 - 11)^2 + 2(x_9 - 10)^2 \\
& + (x_{10} - 7)^2 + 45
\end{aligned} \tag{B2.7.30}$$

subject to

$$\begin{aligned}
&105 - 4x_1 - 5x_2 + 3x_7 - 9x_8 \geq 0 \\
&-10x_1 + 8x_2 + 17x_7 - 2x_8 \geq 0 \\
&8x_1 - 2x_2 - 5x_9 + 2x_{10} + 12 \geq 0 \\
&-3(x_1 - 2)^2 - 4(x_2 - 3)^2 - 2x_3^2 + 7x_4 + 120 \geq 0 \\
&-5x_1^2 - 8x_2 - (x_3 - 6)^2 + 2x_4 + 40 \geq 0 \\
&-x_1^2 - 2(x_2 - 2)^2 + 2x_1 x_2 - 14x_5 + 6x_6 \geq 0 \\
&-0.5(x_1 - 8)^2 - 2(x_2 - 4)^2 - 3x_5^2 + x_6 + 30 \geq 0 \\
&3x_1 - 6x_2 - 12(x_9 - 8)^2 + 7x_{10} \geq 0 \\
&-10.0 \leq x_i \leq 10.0 \qquad i = 1, \ldots, 10.
\end{aligned}$$

Minimum: $f(2.171\,996, 2.363\,683, 8.773\,926, 5.095\,984, 0.990\,654\,8, 1.430\,574, 1.321\,644, 9.828\,726, 8.280\,092, 8.375\,927) = 24.306\,209\,1$. Six (out of eight) constraints are active at the global optimum (all except the last two).

Problem B2.7.4 (Keane's function).

$$f(\boldsymbol{x}) = \left| \frac{\sum_{i=1}^{n} \cos^4(x_i) - 2\prod_{i=1}^{n} \cos^2(x_i)}{\left(\sum_{i=1}^{n} i x_i^2\right)^{1/2}} \right| \tag{B2.7.31}$$

subject to

$$\begin{aligned}
&\prod_{i=1}^{n} x_i > 0.75 \\
&\sum_{i=1}^{n} x_i < 7.5n \\
&0 < x_i < 10 \qquad i = 1, \ldots, n.
\end{aligned}$$

The global maximum of this problem is unknown. The ratio between the size of the feasible search space and the whole search space is approximately 99.97%.

B2.7.4.5 Randomly perturbed functions

In principle, any of the test problems presented here can easily be transformed into a noisy function by adding a normally distributed perturbation according to $\tilde{f} = f + \mathbf{N}(0, \sigma_\delta^2)$. Hammel and Bäck (1994) did so for the sphere model and Rastrigin's function. De Jong (1975) defined a noisy function according to

$$f(\boldsymbol{x}) = \sum_{i=1}^{n} i x_i^4 + \mathbf{N}(0, 1) \tag{B2.7.32}$$

where the variance of the noise term was fixed to a value of one. Minimum: $\mathbf{E}[f(\mathbf{0})] = 0$. Domains of variables for genetic algorithms: $-1.28 \leq x_i \leq 1.28$.

B2.7.5 Royal road functions†

Melanie Mitchell and Stephanie Forrest

Abstract

We describe a class of fitness landscapes called *royal road functions* that isolate some of the features of fitness landscapes thought to be most relevant to the performance of genetic algorithms (GAs). We review experimental results comparing the performance of a GA on an instance of this class with that of three different hill-climbing methods, and we explain why one of the hill climbers, random mutation hill climbing (RMHC), significantly outperforms the GA on this fitness function. We then define an idealized genetic algorithm (IGA) that does explicitly what the GA is thought to do implicitly, and explain why the IGA is significantly faster than RMHC. Our analyses are relevant to understanding how the GA works, on what kinds of landscapes it will work well, and how it may be improved.

An important goal of research on genetic algorithms (GAs) is to understand the class of problems for which GAs are most suited, and, in particular, the class of problems on which they will outperform other search algorithms such as gradient methods. We have developed a class of fitness landscapes—the *royal road* functions (Mitchell *et al* 1992, Forrest and Mitchell 1993)—that isolate some of the features of fitness landscapes thought to be most relevant to the performance of GAs. Our goal in constructing these landscapes is to understand in detail how such features affect the search behavior of GAs and to carry out systematic comparisons between GAs and other search methods.

It has been hypothesized that GAs work by discovering, emphasizing, and recombining high-quality *building blocks* of solutions in a highly parallel manner (Holland 1975, Goldberg 1989b). These ideas are formalized by the *schema theorem* and *building-block hypothesis* (see Section B2.5). The GA evaluates populations of strings explicitly, and at the same time, it is argued, it implicitly estimates, reinforces, and recombines short, high-fitness *schemata*—building blocks encoded as templates, such as 11****** (a template representing all eight-bit strings beginning with two ones).

A simple royal road function, R_1, is shown in figure B2.7.4. R_1 consists of a list of partially specified bit strings (*schemata*) s_i in which '$*$' denotes a wild card (i.e. it is allowed to be either zero or one). A bit string x is said to be an *instance* of a schema s, $x \in s$, if x matches s in the defined (i.e. non-'$*$') positions. The fitness $R_1(x)$ of a bit string x is defined as follows:

$$R_1(x) = \sum_{i=1}^{8} \delta_i(x) o(s_i) \qquad \delta_i(x) = \begin{cases} 1 & \text{if } x \in s_i \\ 0 & \text{otherwise} \end{cases}$$

where $o(s_i)$, the *order* of s_i, is the number of defined bits in s_i. For example, if x is an instance of exactly two of the order-8 schemata, $R_1(x) = 16$. Likewise, $R_1(111\ldots1) = 64$. R_1 is meant to capture

† This work has been supported by the Santa Fe Institute's Adaptive Computation Program, the Alfred P Sloan Foundation (grant B1992-46), and the National Science Foundation (grants IRI-9157644 and IRI-9224912).

one landscape feature of particular relevance to GAs: the presence of fit low-order building blocks that recombine to produce fitter, higher-order building blocks. (A different class of functions, also called *royal road functions*, was developed by Holland and is described by Jones (1995c).)

s_1 = 11111111**;
s_2 = ********11111111**;
s_3 = ****************11111111**;
s_4 = ************************11111111**;
s_5 = ********************************11111111**;
s_6 = **11111111**;
s_7 = **11111111********************;
s_8 = **11111111;

Figure B2.7.4. Royal road function R_1.

The building-block hypothesis implies that such a landscape should lay out a *royal road* for the GA to reach strings of increasingly higher fitnesses. One might also expect that simple hill climbing schemes would perform poorly because a large number of bit positions must be optimized simultaneously in order to move from an instance of a low-order schema (e.g. 11111111**...*) to an instance of a higher-order intermediate schema (e.g. 11111111********11111111**...*). However, the results of our experiments ran counter to both these expectations (Forrest and Mitchell 1993). In these experiments, a simple GA (using fitness-proportionate selection with sigma scaling, single-point crossover, and point mutation—see Chapters C2 and C3) optimized R_1 quite slowly, at least in part because of *hitchhiking*: once an instance of a higher-order schema was discovered, its high fitness allowed the schema to spread quickly in the population, with zeros in other positions in the string hitchhiking along with the ones in the schema's defined positions. This slowed down the discovery of schemata in the other positions, especially those that are close to the highly fit schema's defined positions. Hitchhiking can in general be a serious bottleneck for the GA, and we observed similar effects in several variations of our original GA.

C2, C3

The other hypothesis—that the GA would outperform simple hill climbing on these functions—was also proved wrong. We compared the GA's performance on R_1 (and variants of it) with three different hill-climbing methods: steepest-ascent hill climbing (SAHC), next-ascent hill climbing (NAHC), and random mutation hill climbing (RMHC) (Forrest and Mitchell 1993). These work as follows (assuming that max_evaluations is the maximum number of fitness function evaluations allowed).

Steepest-ascent hill climbing (SAHC):

(i) Choose a string at random. Call this string current_hilltop.
(ii) If the optimum has been found, stop and return it. If max_evaluations has been equaled or exceeded, stop and return the highest hilltop that was found. Otherwise continue to step (iii).
(iii) Systematically mutate each bit in the string from left to right, recording the fitnesses of the resulting strings.
(iv) If any of the resulting strings give a fitness increase, then set current_hilltop to the resulting string giving the highest fitness increase, and go to step (ii).
(v) If there is no fitness increase, then save current_hilltop in a list of all hilltops found and go to step (i).

Next-ascent hill climbing (NAHC):

(i) Choose a string at random. Call this string current_hilltop.
(ii) If the optimum has been found, stop and return it. If max_evaluations has been equaled or exceeded, stop and return the highest hilltop that was found. Otherwise continue to step (iii).
(iii) Mutate single bits in the string from left to right, recording the fitnesses of the resulting strings. If any increase in fitness is found, then set current_hilltop to that increased-fitness string, without evaluating any more single-bit mutations of the original string. Go to step (ii) with the new current_hilltop, but continue mutating the new string starting after the bit position at which the previous fitness increase was found.
(iv) If no increases in fitness are found, save current_hilltop and go to step (i).

Notice that this method is similar to Davis's (1991) *bit-climbing* scheme, in which the bits are mutated in a random order, and current_hilltop is reset to any string having fitness equal to or better than the previous best evaluation.

Random-mutation hillclimbing (RMHC):

(i) Choose a string at random. Call this string best_evaluated.
(ii) If the optimum has been found, stop and return it. If max_evaluations has been equaled or exceeded, stop and return the current value of best_evaluated. Otherwise go to step (iii).
(iii) Choose a locus at random to mutate. If the mutation leads to an equal or higher fitness, then set best_evaluated to the resulting string, and go to step (ii).

Note that in SAHC and NAHC the current string is replaced only if an *improvement* in fitness is found, whereas in RMHC the current string is replaced whenever a string of *equal* or greater fitness is found. This difference allows RMHC to explore *plateaus*, which, as will be seen, produces a large difference in performance.

The results of SAHC and NAHC on R_1 were as expected—while the GA found the optimum on R_1 in an average of ∼60 000 function evaluations, neither SAHC nor NAHC ever found the optimum within the maximum of 256 000 function evaluations. However, RMHC found the optimum in an average of ∼6000 function evaluations—approximately a factor of ten faster than the GA. This striking difference on landscapes originally designed to be royal roads for the GA underscores the need for a rigorous answer to the question posed earlier: 'Under what conditions will a GA outperform other search algorithms, such as hill climbing?'.

To answer this, we first performed a mathematical analysis of RMHC, which showed that the expected number of function evaluations to reach the optimum on an R_1-like function with N blocks of K ones is $\sim 2^K N(\log N + \gamma)$ (where γ is Euler's constant) (Mitchell *et al* 1994; our analysis is similar to that given for a similar problem by Feller (1960, p 210).) We then described and analyzed an *idealized GA* (IGA), a very simple procedure that significantly outperforms RMHC on R_1. The IGA works as follows. On each iteration, a new string is chosen at random, with each bit independently being set to zero or one with equal probability. If a string is found that contains one or more of the desired schemata, that string is saved. When a string containing one or more not-yet-discovered schemata is found, it is crossed over with the saved string in such a way so that the saved string contains all the desired schemata that have been found so far. (Note that the probability of finding a given eight-bit schema in a randomly chosen string is $1/256$.)

This procedure is unusable in practice, because it requires knowing precisely what the desired schemata are. However, the idea behind the IGA is that it does explicitly what the GA is thought to do implicitly, namely identify and sequester desired schemata via reproduction and crossover (*exploitation*) and sample the search space via the initial random population, random mutation, and crossover (*exploration*). We showed that the expected number of function evaluations for the IGA to reach the optimum on an R_1-like function with N blocks of K ones is $\sim 2^K(\log N + \gamma)$, approximately a factor of N faster than RMHC (Mitchell *et al* 1994).

What makes the IGA so much faster than the simple GA and RMHC on R_1? A primary reason is that the IGA perfectly implements the notion of *implicit parallelism* (Holland 1975): each new string is completely independent of the previous one, so new samples are given independently to each schema region. In contrast, RMHC moves in the space of strings by single-bit mutations from an original string, so each new sample has all but one of the same bits as the previous sample. Thus each new string gives a new sample to only one schema region. We ignore the construction time to construct new samples and compare only the number of function evaluations to find particular fitness values. This is because in most interesting GA applications, the time to perform a function evaluation dominates the time required to execute the other parts of the algorithm. For this reason, we assume that the remaining parts of the algorithm take a constant time per function evaluation.

The IGA gives a lower bound on the expected number of function evaluations that the GA will need to solve this problem. It is a lower bound because the IGA is given perfect information about the desired schemata, which is not available to the simple GA. (If it were, there would be no need to run the GA because the problem solution would already be known.)

Independent sampling allows for a speedup in the IGA in two ways: it allows for the possibility of more than one desired schema appearing simultaneously on a given sample, and it also means that there

are no wasted samples, as there are in RMHC. Although the comparison we have made is with RMHC, the IGA will also be significantly faster on R_1 (and similar landscapes) than any hill-climbing method that works by mutating single bits (or a small number of bits) to obtain new samples.

The hitchhiking effects described earlier also result in a loss of independent samples for the real GA. The goal is to have the real GA, as much as possible, approximate the IGA. Of course, the IGA works because it explicitly knows what the desired schemata are; the real GA does not have this information and can only estimate what the desired schemata are by an implicit sampling procedure. However, it is possible for the real GA to approximate a number of the features of the IGA:

- *Independent samples:* The population size has to be sufficiently large, the selection process has to be sufficiently slow, and the mutation rate has to be sufficiently great to ensure that no single locus is fixed at a single value in every string (or even a large majority of strings) in the population.
- *Sequestering desired schemata:* Selection has to be strong enough to preserve desired schemata that have been discovered, but it also has to be slow enough (or, equivalently, the relative fitness of the non-overlapping desirable schemata has to be small enough) to prevent significant hitchhiking on some highly fit schemata, which can crowd out desired schemata in other parts of the string.
- *Instantaneous crossover:* The crossover rate has to be such that the time until a crossover combines two desired schemata is small with respect to the discovery time for the desired schemata.
- *Speedup over RMHC*: The string length (a function of N) has to be large enough to make the N speedup factor significant.

These mechanisms are not all mutually compatible (e.g. high mutation works against sequestering schemata), and thus must be carefully balanced against one another. A discussion of how such a balance might be achieved is given by Holland (1993); some experimental results are given by Mitchell *et al* (1994).

In conclusion, our investigations of a simple GA, RMHC, and the IGA on R_1 and related landscapes are one step towards our original goals—to design the simplest class of fitness landscapes that will distinguish the GA from other search methods, and to characterize rigorously the general features of a fitness landscape that make it suitable for a GA. Our results have shown that it is not enough to invoke the building-block hypothesis for this purpose. Royal road landscapes such as R_1 are not meant to be realistic examples of problems to which one might apply a GA. Rather, they are meant to be idealized problems in which certain features most relevant to GAs are explicit, so that the GA's performance can be studied in detail. Our claim is that, in order to understand how the GA works in general and where it will be most useful, we must first understand how it works and where it will be most useful on simple yet carefully designed landscapes such as these.

References

Ackley D H 1987 *A Connectionist Machine for Genetic Hillclimbing* (Boston, MA: Kluwer)

Altenberg L 1994a The evolution of evolvability in genetic programming *Advances in Genetic Programming* ed K E Kinnear (Cambridge, MA: MIT Press) pp 47–74

——1994b Evolving better representations through selective genome growth *Proc. 1st IEEE Conf. on Evolutionary Computation (Orlando, FL, 1994)* Part 1 (Piscataway, NJ: IEEE) pp 182–7

——1995 The schema theorem and Price's theorem *Foundations of Genetic Algorithms 3 (San Francisco, CA)* ed D Whitley and M D Vose (San Mateo, CA: Morgan Kaufmann) pp 23–49

Anderson P W 1985 Spin glass Hamiltonians: a bridge between biology, statistical mechanics, and computer science *Emerging Synthesis in Science: Proc. Founding Workshops Santa Fe Institute* ed D Pines (Santa Fe, NM: Santa Fe Institute)

Bäck T 1996 *Evolutionary Algorithms in Theory and Practice* (New York: Oxford University Press)

Bäck T and Hoffmeister F 1991 Extended selection mechanisms in genetic algorithms *Proc. 4th Int. Conf. on Genetic Algorithms* ed R K Belew and L B Booker (San Mateo, CA: Morgan Kaufmann) pp 92–9

Bäck T, Rudolph G and Schwefel H-P 1993 Evolutionary programming and evolution strategies: similarities and differences *Proc. 2nd Ann. Conf. on Evolutionary Programming* ed D B Fogel and W Atmar (San Diego, CA: Evolutionary Programming Society) pp 11–22

Bak P, Tang C and Wiesenfeld K 1988 Self-organized criticality *Phys. Rev.* A **38** 364–74

Bergman A, Goldstein D B, Holsinger K E and Feldman M W 1995 Population structure, fitness surfaces, and linkage in the shifting balance process *Genet. Res.* **66** 85–92

Colville A R 1968 *A Comparative Study on Nonlinear Programming Codes* IBM Scientific Center Technical Report 320-2949

Davis L D 1991 Bit-climbing, representational bias, and test suite design *Proc. 4th Int. Conf. on Genetic Algorithms* ed R K Belew and L B Booker (San Mateo, CA: Morgan Kaufmann) pp 18–23

Deb K 1991 *Binary and Floating-point Function Optimization using Messy Genetic Algorithms* Doctoral Dissertation, University of Alabama; IlliGAL Report 91004; *Dissertation Abstracts Int.* **52** 2658B

Deb K and Goldberg D E 1992 Analyzing deception in trap functions *Foundations of Genetic Algorithms 2 (Vail, CO)* ed D Whitley (San Mateo, CA: Morgan Kaufmann) pp 93–108

——1994 Sufficient conditions for arbitrary binary functions *Ann. Math. Artificial Intell.* **10** 385–408

De Jong K A 1975 *An Analysis of the Behaviour of a Class of Genetic Adaptive Systems* PhD Thesis, University of Michigan

Feller W 1960 *An Introduction to Probability Theory and its Applications* 2nd edn (New York: Wiley)

Fletcher R and Powell M J D 1963 A rapidly convergent descent method for minimization *Comput. J.* **6** 163–8

Floudas C A and Pardalos P M 1990 *A Collection of Test Problems for Constrained Global Optimization* (Berlin: Springer)

Fogel D and Beyer H-G 1995 A note on the empirical evaluation of intermediate recombination *Evolutionary Comput.* **3** 491–5

Fontana W, Stadler P F, Bornberg-Bauer E G, Griesmacher T, Hofacker I L, Tacker M, Tarazona P, Weinberger E D and Schuster P 1993 RNA folding and combinatory landscapes *Phys. Rev.* E **47** 2083–99

Forrest S and Mitchell M 1993 Relative building block fitness and the building block hypothesis *Foundations of Genetic Algorithms 2* ed L D Whitley (San Francisco, CA: Morgan Kaufmann) pp 109–26

Garey M R and Johnson D S 1979 *Computers and Intractibility* (San Francisco, CA: Freeman)

Gillespie J H 1984 Molecular evolution over the mutational landscape *Evolution* **38** 1116–29

Goldberg D E 1989a Genetic algorithms and Walsh functions: part I, a gentle introduction *Complex Syst.* **3** 129–52

——1989b *Genetic Algorithms in Search, Optimization, and Machine Learning* (Reading, MA: Addison-Wesley)

——1990 *Construction of High-order Deceptive Functions using Low-order Walsh Coefficients* IlliGAL Report 90002

Goldberg D E, Deb K and Horn J 1992 Massive multimodality, deception, and genetic algorithms *Parallel Problem Solving from Nature II (Brussels)* ed R Manner and B Manderick (Amsterdam: North-Holland) pp 37–46

Goldberg D E, Deb K, Kargupta H and Harik G 1993 Rapid, accurate optimization of difficult problems using messy genetic algorithms *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 56–64

Goldberg D E, Deb K and Korb B 1990 Messy genetic algorithms revisited: nonuniform size and scale *Complex Syst.* **4** 415–44

Goldberg D E, Korb B and Deb K 1989 Messy genetic algorithms: motivation, analysis, and first results *Complex Syst.* **3** 493–530

Grefenstette J J 1993 Deception considered harmful *Foundations of Genetic Algorithms 2 (Vail, CO)* ed D Whitley (San Mateo, CA: Morgan Kaufmann) pp 75–91

Hammel U and Bäck T 1994 Evolution strategies on noisy functions: how to improve convergence properties *Parallel Problem Solving from Nature—PPSN III, Int. Conf. on Evolutionary Computation (Lecture Notes in Computer Science 866)* ed Y Davidor, H-P Schwefel and R Männer (Berlin: Springer) pp 159–68

Hock W and Schittkowski K 1981 *Test Examples for Nonlinear Programming Codes (Lecture Notes in Economics and Mathematical Systems 187)* (Berlin: Springer)

Hoffmeister F and Bäck T 1991 *Genetic Algorithms and Evolution Strategies—Similarities and Differences (Papers on Economics and Evolution 9103)* (Freiburg: The European Study Group on Economics and Evolution)

Holland J H 1975 *Adaptation in Natural and Artificial Systems* (Ann Arbor, MI: University of Michigan Press) (Second edition: 1992 Cambridge, MA: MIT Press)

——1993 *Innovation in Complex Adaptive Systems: Some Mathematical Sketches* Santa Fe Institute Working Paper 93-10-062

Jones T 1995a *Evolutionary Algorithms, Fitness Landscapes and Search* PhD Thesis, University of New Mexico and Santa Fe Institute

——1995b *One Operator, One Landscape* Santa Fe Institute Working Papers 95-02-025

——1995c A description of Holland's royal road function *Evolutionary Comput.* **2** 409–15

Jones T and Forrest S 1995 Fitness distance correlation as a measure of problem difficulty for genetic algorithms *Proc. 6th Int. Conf. on Genetic Algorithms (Pittsburgh, PA)* ed L J Eshelman (San Mateo, CA: Morgan Kaufmann) pp 184–92

Kauffman S A 1989 Adaptation on rugged fitness landscapes ed D Stein (Redwood City, CA: Addison-Wesley) *SFI Studies in the Sciences of Complexity, Lecture Volume I* pp 527–618

——1993 *The Origins of Order: Self-Organization and Selection in Evolution* (New York: Oxford University Press)

Kauffman S A and Levin S 1987 Towards a general theory of adaptive walks on rugged landscapes *J. Theor. Biol.* **128** 11–45

Kargupta H, Deb K and Goldberg D E 1992 Ordering genetic algorithms and deception *Parallel Problem Solving from Nature II (Brussels)* ed R Männer and B Manderick (Amsterdam: North-Holland) pp 47–56

Kingman J F C 1978 A simple model for the balance between selection and mutation *J. Appl. Probability* **15** 1–12

——1980 *Mathematics of Genetic Diversity* (Philadelphia, PA: Society for Industrial and Applied Mathematics) p 15
Liepins G E and Vose M D 1990 Representational issues in genetic optimization *J. Exp. Theor. Artficial Intell.* **2** 4–30
Manderick B, de Weger M and Spiessens P 1991 The genetic algorithm and the structure of the fitness landscape *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA)* ed R Belew and L B Booker (San Mateo, CA: Morgan Kaufmann) pp 143–50
Mason A J 1991 Partition coefficients, static deception and deceptive problems *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA)* ed R K Belew and L B Booker (San Mateo, CA: Morgan Kaufmann) pp 210–4
Mathias K and Whitley D 1992 Genetic operators, the fitness landscape and the traveling salesman problem *Parallel Problem Solving from Nature (Brussels)* vol 2, ed R Männer and B Manderick (Amsterdam: Elsevier) pp 219–28
Maynard Smith J 1970 Natural selection and the concept of a protein space *Nature* **225** 563–4
Michalewicz Z 1995 Genetic algorithms, nonlinear optimization, and constraints *Proc. 6th Int. Conf. on Genetic Algorithms* ed L Eshelman (San Francisco, CA: Morgan Kaufmann) pp 151–8
Michalewicz Z, Logan T D and Swaminathan S 1994 Evolutionary operators for continuous convex parameter spaces *Proc. 3rd Ann. Conf. on Evolutionary Programming* ed A V Sebald and L J Fogel (Singapore: World Scientific) pp 84–97
Mitchell M, Forrest S and Holland J H 1992 The royal road for genetic algorithms: fitness landscapes and GA performance *Toward a Practice of Autonomous Systems: Proc. 1st Eur. Conf. on Artificial Life (Paris, 1991)* ed F J Varela and P Bourgine (Cambridge, MA: MIT Press) pp 245–54
Mitchell M, Holland J H and Forrest S 1994 When will a genetic algorithm outperform hill climbing? *Advances in Neural Information Processing Systems 6* ed J D Cowan, G Tesauro and J Alspector (San Francisco, CA: Morgan Kaufmann) pp 51–8
Mühlenbein H and Schlierkamp-Voosen D 1993 Predictive models for the breeder genetic algorithm *Evolutionary Comput.* **1** 25–49
Oliver I M, Smith D J and Holland J R C 1987 A study of permutation crossover operators on the traveling salesman problem *Genetic Algorithms and their Applications: Proc. 2nd Int. Conf. on Genetic Algorithms (Pittsburgh, PA)* ed J J Grefenstette (Hillsdale, NJ: Erlbaum) pp 224–30
Perelson A S and Macken C A 1995 Protein evolution on partially correlated landscapes *Proc. Natl Acad. Sci. USA* **92** 9657–61
Press W H, Teukolsky S A, Vetterling W T and Flannery B P 1992 *Numerical Recipes in C: the Art of Scientific Computing* 2nd edn (Cambridge: Cambridge University Press) pp 178–80, 300–4
Rosenbrock H H 1960 An automatic method for finding the greatest or least value of a function *Comput. J.* **3** 175–84
Schwefel H-P 1977 *Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie (Interdisciplinary Systems Research 26)* (Basel: Birkhäuser)
——1988 Evolutionary learning optimum—seeking on parallel computer architectures *Proc. Int. Symp. on Systems Analysis and Simulation 1988, I: Theory and Foundations* ed A Sydow, S G Tzafestas and R Vichnevetsky (Berlin: Academic) pp 217–25
——1995 *Evolution and Optimum Seeking (Sixth-Generation Computer Technology Series)* (New York: Wiley)
Stadler P F and Happel R 1992 Correlation structure of the landscape of the graph-bipartition problem *J. Phys. A.: Math. Gen.* **25** 3103–10
——1995 *Random Field Models for Fitness Landscapes* Santa Fe Institute Working Papers 95-07-069
Stadler P F and Schnabl W 1992 The landscape of the traveling salesman problem *Phys. Lett.* **161A** 337–44
Thompson R K and Wright A H 1996 Additively decomposable fitness functions, at press
Törn A and Žilinskas A 1989 *Global Optimization (Lecture Notes in Computer Science 350)* (Berlin: Springer)
Van Valen L 1973 A new evolutionary theory *Evolutionary Theory* **1** 1
Weinberger E D 1990 Correlated and uncorrelated fitness landscapes and how to tell the difference *Biol. Cybernet.* **63** 325–36
——1991 Local properties of Kauffman's N–k model, a tuneably rugged energy landscape *Phys. Rev.* A **44** 6399–413
——1996 *NP Completeness of Kauffman's N–k Model, a Tuneable Rugged Fitness Landscape* Santa Fe Institute Working Papers 96-02-003, first circulated in 1991
Whitley D and Mathias K and Rana S and Dzubera J 1995 Building better test functions *Proc. 6th Int. Conf. on Genetic Algorithms* ed L Eshelman (San Francisco, CA: Morgan Kaufmann) pp 239–46
Whitley D 1991 Fundamental principles of deception in genetic search *Foundations of Genetic Algorithms (Bloomington, IN)* ed G J E Rawlins (San Mateo, CA: Morgan Kaufmann) pp 221–41
Wirth N 1975 *Algorithms + Data Structures = Programs* (Englewood Cliffs, NJ: Prentice-Hall)
Wright S 1932 The roles of mutation, inbreeding, crossbreeding, and selection in evolution *Proc. 6th Int. Congr. on Genetics (Ithaca, NY, 1932)* vol 1, ed D F Jones (Menasha, WI: Brooklyn Botanical Gardens) pp 356–66

B2.8 Probably approximately correct (PAC) learning analysis

Johannes P Ros

Abstract

A class of genetic algorithms (GAs) for learning bounded Boolean conjuncts and disjuncts is presented and analyzed in the context of computational learning theory. Given any reasonable recombination operator, and any confidence and accuracy level, the results in this article provide the number of generations and the size of the population sufficient for the genetic algorithm to become a polynomial-time *probably approximately correct* (PAC) learner for the target classes k-CNF (conjunctive normal form) and k-DNF (disjunctive normal form) Boolean formulas of ν variables.

B2.8.1 Introduction

In this article, a class of genetic algorithms (GAs) for learning bounded Boolean conjuncts and disjuncts is presented and analyzed in the context of computational learning theory. Given any reasonable *recombination operator*, and any confidence and accuracy level, the results in this article provide the C3.3 number of generations and the size of the population sufficient for the GA to become a polynomial-time *probably approximately correct* (PAC) learner for the target classes k-CNF (conjunctive normal form) and k-DNF (disjunctive normal form) Boolean formulas of ν variables. The set of k-CNF formulas comprises all Boolean functions that can be expressed as the conjunction of *clauses*, where each clause is a disjunct of at most k literals. Similarly, the set of k-DNF formulas is all Boolean functions that can be expressed as the disjunction of *terms*, where each term is a conjunct of at most k literals.

The results in this article are based on the work on PAC learning analysis by Ros (1992), where further details can be found. To enhance the presentation of these results, we have ignored the constants (which can be obtained from lemmas A.16 and A.17 of Ros (1992)), and have used the O-notation to denote asymptotic upper bounds, the Ω-notation to denote asymptotic lower bounds, and the Θ-notation to denote asymptotic tight bounds (Cormen *et al* 1991).

B2.8.2 Computational learning theory

Computational learning theory deals with theoretical issues in machine learning from a quantitative point of view. The goal is to produce practical learning algorithms in non-trivial situations based on assumptions that capture the essential aspects of the learning process. In this context, the learning model consists of a world W of concepts, a learning machine M, and an oracle that presents M with a sequence of labeled examples (e.g. strings of bits) from world W. For any concept c from W, it is M's task, after having received a sufficiently large sample from the oracle, to produce a hypothesis that adequately describes c, assuming such a hypothesis exists.

The PAC model of computational learning theory was founded by Valiant in 1984 when he introduced a new formal definition of concept learning, the *distribution-free learnability model* (Valiant 1984). In this model, the learning algorithm produces with high probability an accurate description of the target concept within polynomial time in the number of training examples and the size of the smallest possible representation for the target concept. The training examples are independently selected at random from

unknown (but fixed) probability distributions. The lack of prior knowledge about these distributions justifies the term *distribution free*. The hypotheses produced under such a model are PAC, since with high probability $(1-\delta)$ they represent ε-close approximations to the target concepts (where $\varepsilon, \delta \in \Re^{(0,1)}$).

There are a number of aspects of the PAC model that are notably different from other formal learning paradigms. First, the learner is allowed to *approximate* the target concept rather than to identify it exactly. For example, the inductive inference of classes of recursive functions typically requires exact identification. Second, the model insists on polynomial-time learning algorithms, which is necessary for practical learning systems. Third, the probability distribution over the training examples is *not* a parameter of the model: there is no prior knowledge about this distribution available to the learner. This contrasts with certain statistical pattern recognition techniques where the input distributions are sometimes restricted to certain classes. Finally, the PAC model is not restricted to certain knowledge representations: it treats the representation of hypothesis spaces as a parameter.

The performance of a PAC learner is measured by its sample complexity (i.e. number of training examples) and its computational complexity (i.e. running time). Clearly, the computational complexity is bounded from below by the sample complexity. While for some classes the computational complexity is close to the sample complexity, the complexity of computing a hypothesis from a sample often dominates the total running time. Indeed, for certain hypothesis spaces the computational complexity is believed to be exponential in l, the maximum size of an example.

For finite hypothesis spaces, a simple counting argument provides an upper bound for the sample complexity: $\lceil (1/\varepsilon)\ln(|H_l|/\delta)\rceil$, where ε is the desired accuracy level, δ is the desired confidence level and $|H_l|$ denotes the number of possible hypotheses. For all those concept classes where the counting method does not yield optimal results or does not apply (e.g. infinite classes), the sample complexity may be obtained via the Vapnik–Chervonenkis (VC) dimension of the hypothesis space (Vapnik and Chervonenkis 1971, Blumer *et al* 1989).

B2.8.3 The genetic probably approximately correct learner

This section describes the population, the fitness function, and the genetic plan of the genetic PAC learner for the classes of k-CNF and k-DNF Boolean formulas. These formulas are the largest Boolean classes known to be PAC learnable in polynomial time in v (number of Boolean variables), $1/\varepsilon$, and $1/\delta$. In particular, Valiant (1984) showed that k-CNF is PAC learnable in polynomial time using only positive examples, and k-DNF using only negative examples. Hence, our GA will use positive examples for learning k-CNF, and negative examples for k-DNF.

B2.8.3.1 The population

The GA maintains a population of *bit strings* of length l that represent the sets of k-DNF and k-CNF Boolean formulas over v Boolean variables. More specifically, the set of k-DNF formulas is represented by $D_{v,k} = \{0,1\}^l$, where $l = O(v^k)$ is the number of terms, and every bit represents the presence or absence of a term. For example, $D_{3,2}$ needs 18 bits to represent all its possible terms: C1.2

$$x_1, x_2, x_3, \bar{x}_1, \bar{x}_2, \bar{x}_3, x_1x_2, x_1x_3, x_2x_3, \bar{x}_1x_2, \bar{x}_1x_3, \bar{x}_2x_3, x_1\bar{x}_2, x_1\bar{x}_3, x_2\bar{x}_3, \bar{x}_1\bar{x}_2, \bar{x}_1\bar{x}_3, \bar{x}_2\bar{x}_3.$$

Similarly, the set of k-CNF formulas is represented by $C_{v,k} = \{0,1\}^l$, where $l = O(v^k)$ is the number of clauses. Since the main result in the next section applies to both $C_{v,k}$ and $D_{v,k}$ (by the duality principle), we define $\Gamma \equiv \{0,1\}$, and use Γ^l to denote either class, with l being the number of *attributes* (clauses or terms).

The size of the population is denoted by μ, and is fixed during the entire learning process.

B2.8.3.2 The fitness function

The fitness function F takes three arguments: representation $r \in \Gamma^l$, training example $e \in \{0,1\}^v$, and the classification of this example: $\copyright \in \{+,-\}$. The total credit is the sum of the credit values that are generated by each individual attribute in the formula. These credit values are positive integers and take the following values:

- G_{00}, if attribute is absent from r, and e shows that it must be absent

- G_{01}, if attribute is absent from r, but e shows that it may have to be present
- G_{10}, if attribute is present in r, but e shows that it must be absent
- G_{11}, if attribute is present in r, and e shows that it may have to be present.

For example, suppose that out of five attributes (i.e. $l = 5$), a_1 and a_3 are present in representation r, and all others are absent (i.e. $r = 10100$). Suppose further that e and © show that attributes a_3 and a_5 must be absent and that a_1, a_2, and a_4 may have to be present. Then, r receives a total credit of $F(r, e, ©) = G_{11} + G_{01} + G_{10} + G_{01} + G_{00}$, for $a_1, \ldots, a_5$, respectively.

We say that $F(r, e, ©)$ is a *productive fitness function* if the following credit relations are satisfied: $1 \leq G_{10} < G_{01} < G_{11} = G_{00}$.

B2.8.3.3 The crossover operator

For any $p \geq 2$, the *crossover operator* χ is a (possibly randomized) function from $(\Gamma^l)^p$ to Γ^l that maps p parental structures $w_1, \ldots, w_p$ into one offspring structure w (i.e. $w \leftarrow \chi(w_1, \ldots, w_p)$), by having the parents donate their symbols to the offspring structure. The *shuffle factor* of χ, $\mathrm{SF}(\chi)$, is defined as the smallest probability of the event that any two positions in the offspring structure receive their symbols from two *different* parental structures. The shuffle factor of a crossover operator characterizes its *disruptiveness*: a larger shuffle factor corresponds to a more disruptive operator. For two parents (i.e. $p = 2$) the uniform crossover operator has a shuffle factor of 0.5, since for any two positions in the offspring structure the probability that they receive their symbol from two different parents is 0.5. Similarly, the one-point crossover operator has a shuffle factor of only $\Theta(1/l)$ (where l is the length of the structure), since for any two positions in the structure the probability that the cut will be made between these two positions can be as small as $1/l$ (or of that magnitude, depending on the implementation). A shuffle factor with the value of unity can be obtained with a *deterministic crossover operator* that, on input of l parental structures, produces an offspring structure of length l in which each bit is copied from a different parent. Therefore, this operator is the most disruptive crossover within our framework, and produces the most efficient GA in terms of the population size and the number of generations, as is shown by the corollary in the next section.

C3.3.1

B2.8.3.4 The genetic plan

The main result is based on the genetic plan at the top of the next page. For all $\nu, k \in \aleph^+$, let Γ^l denote the hypothesis class k-CNF (k-DNF) such that $l \geq 2$. Then, for all target functions $f \in k$-CNF (k-DNF), all accuracy parameters $\varepsilon \in \Re^{(0,1)}$, and all confidence parameters $\delta \in \Re^{(0,1)}$, the genetic plan at the top of the next page computes the function $\mathrm{GF}_{\Gamma,\chi,m,F,\mu}(\nu, l, \varepsilon, \delta)$, where m is the number of generations.

The population is initialized by selecting uniformly at random elements from the set Γ^l (statement 1). Then, for m generations, a positive or negative training example is obtained from the oracle (statement 2), and a new population is formed by assigning credit and applying *proportional selection* and the crossover operator to the current population (statement 3). After m generations, the final hypothesis is selected at random from the last population (statements 4, 5).

C2.2

The probability distribution function (pdf) $D_{B_t} \in \boldsymbol{D}_{\Gamma^l}$ provides the probability that formula $r \in \Gamma^l$ occurs in population B_t. In other words, $D_{B_t}(r) \equiv \mathbf{Pr}\left[\mathrm{RAND}(D_{B_t}) = r\right]$. For each offspring structure, the genetic plan randomly draws $p \geq 2$ parents under the proportional selection scheme before applying the crossover operator. Proportional selection implies that the probability of each individual r in the population is weighted based on the amount of credit it receives from fitness function $F(r, e, ©)$ relative to the total amount of credit as collected by the entire population, which is characterized by pdf $D_{B_t,F,e,©}(r)$. On top of that, the probability of producing offspring structure r is also determined by the crossover operator, which is characterized by pdf $D_{B_t,F,e,©,\chi}(r)$:

$$D_{B_t,F,e,©,\chi}(r) \equiv \mathbf{Pr}\left[\chi(\mathrm{RAND}_1(D_{B_{t-1},F,e,©}), \ldots, \mathrm{RAND}_p(D_{B_{t-1},F,e,©})) = r\right]. \tag{B2.8.1}$$

B2.8.4 Main result

This section contains the main result, which provides an upper bound on the size of the population and the number of generations for the GA to be a polynomial-time PAC learner for the target classes k-CNF and k-DNF.

function $\mathrm{GF}_{\Gamma,\chi,m,F,\mu}$
Input: $\nu, l, \varepsilon, \delta$
Output: r: an individual (representation) selected at random from final population
{B_t *denotes the* t*-th population*}
{$B_t(j)$ *denotes the* j*-th member of the* t*-th population*}
{$D_{\text{uniform}} \in \boldsymbol{D}_{\Gamma^l}$ *is the uniform distribution over elements in* Γ^l}
{$D_{B_t} \in \boldsymbol{D}_{\Gamma^l}$ *is a pdf over elements in* Γ^l *based on population* B_t}
begin
{*initialize the first population*}
for $j \leftarrow 1$ to μ **do**
1 $B_0(j) \leftarrow \mathrm{RAND}(D_{\text{uniform}})$;
od
{*apply genetic operators to* m *populations*}
for $t \leftarrow 1$ to m **do**
{*obtain a* ν*-bit positive example* (© = '+') *if* Γ^l *represents* k*-CNF formulas*}
{*obtain a* ν*-bit negative example* (© = '−') *if* Γ^l *represents* k*-DNF formulas*}
2 $e \leftarrow \mathrm{ORACLE}^{\nu}$;
for $j \leftarrow 1$ to μ **do** {*obtain* B_t *by applying credit, selection, and crossover*†}
3 $B_t(j) \leftarrow \mathrm{RAND}(D_{B_{t-1},F,e,©,\chi})$; {*see equation (B2.8.1)*}
od
od
{*select final hypothesis at random from final population*}
4 $r \leftarrow \mathrm{RAND}(D_{B_m})$
5 **output**(r)**;**
end.

Theorem. For all $\nu, k \in \aleph^+$, let Γ^l represent the hypothesis space for k-CNF (k-DNF) formulas where l, the number of attributes, is polynomial in the number of Boolean variables ν. Then, for all target functions $f \in k$-CNF (k-DNF), all accuracy parameters $\varepsilon \in \Re^{(0,1)}$ (such that $\varepsilon/l < 0.25$), and all confidence parameters $\delta \in \Re^{(0,1)}$, let G be a GA that computes the function $\mathrm{GF}_{\Gamma,\chi,m,F,\mu}(\nu, l, \varepsilon, \delta)$, where χ is the crossover operator such that $0 < \mathrm{SF}(\chi) \leq 1$, m is the number of generations, F is a productive fitness function with fitness ratios $a \equiv G_{11}/(G_{11} - G_{01})$ and $b \equiv G_{00}/(G_{00} - G_{10})$, and μ is the size of the population. Define $\omega \equiv 1 - \mathrm{SF}(\chi)$.

(i) If $\omega \geq \varepsilon/[l^2 \ln(4l/\delta)]$ (such that $1/(1-\omega)$ is polynomial in ν, $1/\varepsilon$, and $1/\delta$), and if

$$m = \Omega\left(\frac{l^4\omega\ln(l/\delta)}{\varepsilon^2(1-\omega)}\left(\ln\frac{l^5\omega\ln(l/\delta)}{\varepsilon^2\delta(1-\omega)}\right)^2\right) \qquad a = \Theta\left(\frac{l^3\omega(\ln(lm/\delta))^2}{\varepsilon^2(1-\omega)}\right) \tag{B2.8.2}$$

$$b = \Theta\left(\frac{l^2\omega\ln(lm/\delta)}{\varepsilon(1-\omega)}\right) \qquad \mu = \Omega\left(\frac{l^{11}\omega^2(\ln(lm/\delta))^5}{\varepsilon^4\delta^3(1-\omega)^2}\right) \tag{B2.8.3}$$

then G is a (polynomial-time) PAC learner for the classes k-CNF and k-DNF.

(ii) If $0 < \omega < \varepsilon/[l^2 \ln(4l/\delta)]$ (such that $1/\omega$ is polynomial in ν, $1/\varepsilon$, and $1/\delta$), and if

$$m = \Omega\left(\frac{1}{\omega\ln(l/\delta)}\left(\ln\frac{l}{\delta\omega\ln(l/\delta)}\right)^2\right) \qquad a = \Theta\left(\frac{(\ln(lm/\delta))^2}{l\omega(\ln(l/\delta))^2}\right) \tag{B2.8.4}$$

$$b = \Theta\left(\frac{\ln(lm/\delta)}{\ln(l/\delta)}\right) \qquad \mu = \Omega\left(\frac{l^3(\ln(lm/\delta))^5}{\delta^3\omega^2(\ln(l/\delta))^4}\right) \tag{B2.8.5}$$

then G is a (polynomial-time) PAC learner for the classes k-CNF and k-DNF.

(iii) For all rational numbers $b > 1$, if

$$m = \Omega\left(\frac{l^2b^2\ln(l/\delta)}{\varepsilon(b-1+1/l)}\ln\frac{l^3b^2\ln(l/\delta)}{\varepsilon\delta(b-1+1/l)}\right) \qquad a = \Theta\left(\frac{lb^2\ln(lm/\delta)}{\varepsilon(b-1+1/l)}\right) \tag{B2.8.6}$$

† Mutation is not considered here, since a fixed mutation rate is not expected to significantly affect the main result (Ros 1992, Chapter 10).

$$0 \leq \omega \leq \frac{b(b-1)}{b(b-1)+4la} \qquad \mu = \Omega\left(\frac{l^7 b^4 (\ln(lm/\delta))^3 (b/(b-1))^{3(l+1)/l}}{\varepsilon^2 \delta^3 (b-1+1/l)^2}\right) \tag{B2.8.7}$$

then G is a (polynomial-time) PAC learner for the classes k-CNF and k-DNF.

Proof. The proof of this theorem can be found in the dissertation of Ros (1992).

Corollary. If $\omega = 0$ and

$$m = \Omega\left(\frac{l^2 \ln(l/\delta)}{\varepsilon} \ln \frac{l^3 \ln(l/\delta)}{\varepsilon\delta}\right) \qquad a = \Theta\left(\frac{l \ln(lm/\delta)}{\varepsilon}\right) \tag{B2.8.8}$$

$$b = 1 + \Theta(1) \qquad \mu = \Omega\left(\frac{l^7 (\ln(lm/\delta))^3}{\varepsilon^2 \delta^3}\right) \tag{B2.8.9}$$

then G is a (polynomial-time) PAC learner for the classes k-CNF and k-DNF.

B2.8.5 Discussion

The main problem for any GA is to prevent its population from *converging prematurely* to a small set of genotypically identical individuals. Training examples that will point out the fallacy of these dominating individuals may arrive too late due to their low but yet significant probability (i.e. greater than ε).

One way to limit premature convergence is to reduce the amount of credit given to well performing individuals relative to the amount given to weaker ones (which is obtained by increasing the fitness ratios a and b). However, this slows down the overall rate of growth, which means that more generations are necessary to accomplish the same amount of total growth. In addition, by leveling the relative credit values, selection noise may become more significant unless the population size increases.

It turns out that a better way to avoid premature convergence is to improve the effectiveness of the crossover operator, which can be accomplished by increasing the operator's shuffle factor. By definition, a larger shuffle factor increases the chances for every bit of being exchanged. This makes it more difficult for any structure to dominate the population prematurely, because it will be more likely that the crossover operator will quickly break the dominating structures up and redistribute their (well performing) parts among other members of the population.

Parts (i) and (iii) of the theorem show the effect of the various crossover operators. According to (i), the fitness ratios, the number of generations, and the population size decrease if the crossover operator is made more disruptive, all other things being equal. We see a similar behavior in (iii), except for the fact that ω does not appear explicitly in the expressions. Instead, one is able to manipulate the upper bound of ω with fitness ratio b: if $b \to \infty$, ω's upper bound increases (weakening the crossover operator), as does the fitness ratio a, the number of generations, and the population size. On the other hand, if $b \to 1$, ω's upper bound decreases (strengthening the crossover operator), and so do the other variables. Part (ii) of the theorem covers the case where (i) and (iii) do not apply.

As mentioned in section B2.8.3.3, the corollary shows that the l-parental deterministic crossover operator obtains the best results within our framework in terms of the number of generations and the size of the population.

Finally, the analytical tools that were developed to obtain the above results should also carry over to other applications of GAs (e.g. function optimization).

References

Blumer A, Ehrenfeucht A, Haussler D and Warmuth M K 1989 Learnability and the Vapnik–Chervonenkis dimension *J. ACM* **36** 929–65

Cormen T H, Leiserson C E and Rivest R L 1991 *Introduction to Algorithms* (New York: McGraw-Hill)

Ros J P 1992 *Learning Boolean Functions with Genetic Algorithms: a PAC Analysis* Doctoral Dissertation, University of Pittsburgh

Valiant L G 1984 A theory of the learnable *Commun. ACM* **27** 1134–42

Vapnik V N and Chervonenkis A Y 1971 On the uniform convergence of relative frequencies of events to their probabilities *Theor. Prob. Appl.* **16** 264–80

B2.9 Limitations of evolutionary computation methods

Kalyanmoy Deb

Abstract

Evolutionary computation (EC) algorithms are new yet intriguing. Over the years, EC methods have enjoyed widespread application in various problems of science, engineering, and commerce. Because of their diverse applications, they may seem to be a panacea to every problem, but, as for other search and optimization methods, there are limitations to these methods. In this section, we mention some of these limitations.

In the recent past, evolutionary computation (EC) algorithms have been applied to various search and optimization problems with much success. However, like other traditional search and optimization methods they also have some limitations. The limitations mainly come from the improper choice of EC parameters such as the *population size*, *crossover* and *mutation* probabilities, and selection pressure. These methods are not expected to work on arbitrary problems with an arbitrary parameter setting. Thus, to solve a problem efficiently, the users must be aware of the studies related to appropriate parameter choice such as parent and children population sizes, operator probabilities, and representational issues (Bäck *et al* 1993, Goldberg *et al* 1993a, Rudolph 1994). Some of these guidelines are outlined in various chapters of this handbook. Unless these guidelines are properly understood, EC methods may not be used efficiently. To illustrate, let us choose a *genetic algorithm* (GA) application to a simple, bitwise linear, one-max problem of counting 1s (Goldberg *et al* 1993b). With a reasonable population size of 160, string length of 30, crossover probability of 0.3, mutation probability of zero, and *tournament selection* with $s = 5$, the simple tripartite GAs (with selection, crossover, and mutation) could not find the optimal solution in 100 different simulations in a reasonable number of function evaluations. This example is cited not to discourage the readers from using GAs or any other EC methods, but to highlight the fact that, like traditional methods, these methods are also not expected to work successfully with any arbitrary parameter setting. According to the analysis outlined elsewhere (Rudolph 1994), GAs with a small mutation probability would be able to solve the above problem. Thus, the users either choose the parameters according to the guidelines suggested in the literature or perform multiple simulations each beginning with a different initial populations to justify the working of the algorithm.

E1.1, C3.3, C3.2

B1.2

C2.3

Since most EC operators are stochastic in nature, their performance largely depends on the chosen *random number generator*. The user must ensure the randomness of the numbers generated by the random number generator using various measures (Knuth 1981). With a biased random number generator, the stochasticity in the operators will be lost and EC methods may not work properly. Although this is not a limitation of the EC methods *per se*, their successful working assumes the use of a proper random number generator.

E2.1.2

The recombination and mutation operators used in most EC studies are generic in nature so that they can be applied to a wide variety of problems. No gradient or special problem knowledge is usually required in the working of these algorithms. This flexibility has allowed EC methods to be successfully applied to multimodal, complex problems, where most traditional methods are largely unsuccessful. However, this flexibility does not come without any extra cost. Since no gradient information or problem knowledge is used, EC methods may require comparatively more function evaluations than classical search and optimization methods in solving simple, differentiable, unimodal functions. Therefore, it may not be advantageous to apply the canonical EC methods to such simple functions. There exist some classical algorithms with tailored heuristics which are designed to solve a specific class of problems efficiently.

Since problem information is not used in the search process, the canonical EC methods may not be efficient as far as the computational effort is concerned. However, if *knowledge-augmented operators* and/or *hybrid techniques* are used their performance may be comparable to that of the classical methods. Thus, users must be aware of the problem domains where EC methods are advantageous compared to their traditional counterparts.

C3.4.2 D

Since most EC methods require an objective function to evaluate a solution, this may cause some difficulty in using them in modeling problems where the functional relationship among the input and output parameters is not clear. However, the genetic programming technique can find a functional relationship between a given set of input and output data, when all necessary functions and terminals to arrive at the model are specified at the time of creating the initial population. Although there exist some studies where important subfunctions needed to solve the problem can be evolved from basic functions and terminals during the search process, the computational effort needed to solve the problem may be enormous.

References

Bäck T, Rudolph G and Schwefel H-P 1993 Evolutionary programming and evolution strategies: similarities and differences *Proc. 2nd Ann. Conf. on Evolutionary Programming (San Diego, CA)* ed D B Fogel and W Atmar (La Jolla, CA: Evolutionary Programming Society) pp 11–22

Knuth D E 1981 *The Art of Computer Programming* vol 2 (Reading, MA: Addison-Wesley)

Goldberg D E, Deb K and Clark J 1993a Genetic algorithms, noise, and the sizing of populations *Complex Syst.* **6** 333–62

Goldberg D E, Deb K and Theirens D 1993b Toward a better understanding of mixing in genetic algorithms *J. Soc. Instrum. Control Eng.* **32** 10–16

Rudolph G 1994 Convergence analysis of canonical genetic algorithms *IEEE Trans. Neural Networks* **NN-5** 96–101

C1.1 Introduction

Kalyanmoy Deb

Abstract

The structure of a solution vector in any search and optimization problem depends on the underlying problem. In some problems, a solution is a real-valued vector specifying dimensions to the problem's key parameters. In some other problems, a solution is a strategy or an algorithm for achieving a task. As the nature of solutions varies from problem to problem, a solution for a particular problem is also possible to represent in a number of different ways. For example, the decision variables in an engineering optimal design problem can be represented either as a vector of real numbers or as a vector of real and integer numbers, depending on the availability of design components and the choice of the designer. Moreover, the search algorithms are usually efficient in using a particular representation and not so efficient in using other types of representation. Thus, it becomes important to choose the representation that is most suitable for the chosen search algorithm. In this sense, the representation of a solution is one of the important aspects in the successful working of any search algorithm including evolutionary algorithms. In this section, we briefly discuss some of the important representations used in evolutionary computation (EC) studies. Later, we also discuss a mixed-representation scheme for handling mixed-integer programming problems.

C1.1.1 Solutions and representations

Every search and optimization algorithm deals with solutions, each of which represents an instantiation of the underlying problem. Thus, a solution must be such that it can be completely realized in practice; that is, either it can be fabricated in a laboratory or in a workshop or it can be used as a control strategy or it can be used to solve a puzzle, and so on. In most engineering problems, a solution is a real-valued vector specifying dimensions to the key parameters of the problem. In control system problems, a solution is a time- or frequency-dependent functional variation of key control parameters. In game playing and some artificial-intelligence-related problems, a solution is a strategy or an algorithm for solving a particular task. Thus, it is clear that the meaning of a solution is inherent to the underlying problem.

As the structure of a solution varies from problem to problem, a solution of a particular problem can be represented in a number of ways. Usually, a search method is most efficient in dealing with a particular representation and is not so efficient in dealing with other representations. Thus, the choice of an efficient representation scheme depends not only on the underlying problem but also on the chosen search method. The efficiency and complexity of a search algorithm largely depends on how the solutions have been represented and how suitable the representation is in the context of the underlying search operators. In some cases, a difficult problem can be made simpler by suitably choosing a representation that works efficiently with a particular algorithm.

In a classical search and optimization method, all decision variables are usually represented as vectors of real numbers and the algorithm works on one vector of solution to create a new vector of solution (Deb 1995, Reklaitis *et al* 1983). Different EC methods use different representation schemes in their search process. *Genetic algorithms* (GAs) have been mostly used with a binary string representing the decision variables. *Evolution strategy* and *evolutionary programming* studies have used a combination of real- B1.2 B1.3, B1.4

valued decision variables and a set of strategy parameters as a solution vector. In *genetic programming*, a solution is a LISP code representing a strategy or an algorithm for solving a task. In permutation problems solved using an EC method, a series of node numbers specifying a complete permutation is commonly used as a solution. In the following subsection, we describe a number of important representations used in EC studies. B1.5.1

C1.1.2 Important representations

In most applications of GAs, decision variables are coded in binary strings of 1s and 0s. Although the variables can be integer or real valued, they are represented by binary strings of a specific length depending on the required accuracy in the solution. For example, a real-valued variable x_i bounded in the range (a, b) can be coded in five-bit strings with the strings (00000) and (11111) representing the real values a and b, respectively. Any of the other 30 strings represents a solution in the range (a, b). Note that, with five bits, the maximum attainable accuracy is only $(b - a)/(2^5 - 1)$. We shall discuss the binary coding further in Section C1.2. Although binary string coding has been most popular in GAs, a number of researchers prefer to use *Gray* coding to eliminate the Hamming cliff problem associated with binary coding (Schaffer *et al* 1989). In Gray coding, the number of bit differences between any two consecutive strings is one, whereas in binary strings this is not always true. However, as in the binary strings, even in Gray-coded strings a bit change in any arbitrary location may cause a large change in the decoded integer value. Moreover, the decoding of the Gray-coded strings to the corresponding decision variable introduces an artificial nonlinearity in the relationship between the string and the decoded value. C1.2

The coding of the variables in string structures make the search space discrete for GA search. Therefore, in solving a continuous search space problem, GAs transform the problem into a discrete programming problem. Although the optimal solutions of the original continuous search space problem and the derived discrete search space problem may be marginally different (with large string lengths), the obtained solutions are usually acceptable in most practical search and optimization problems. Moreover, since GAs work with a discrete search space, they can be conveniently used to solve discrete programming problems, which are usually difficult to solve using traditional methods.

The coding of the decision variables in strings allows GAs to exploit the similarities among various strings in a population to guide the search. The similarities in string positions are represented by ternary strings (with 1, 0, and $*$, where a $*$ matches a 1 or a 0) known as schema. The power of GA search is considered to lie in the implicit parallel schema processing, a matter which is discussed in detail in Section B2.5. B2.5

Although string codings have been mostly used in GAs, there have been some studies with direct *real-valued* vectors in GAs (Deb and Agrawal 1995, Chaturvedi *et al* 1995, Eshelman and Schaffer 1993, Wright 1991). In those applications, decision variables are directly used and modified genetic operators are used to make a successful search. A detailed discussion of the real-valued vector representations is given in Section C1.3. C1.3

In the evolution strategy (ES) and evolutionary programming (EP) studies, a natural representation of the decision variables is used where a real-valued solution vector is used. The numerical values of the decision variables are immediately taken from the solution vector to compute the objective function value. In both ES and EP studies, the crossover and mutation operators are used variable by variable. Thus, the relative positioning of the decision variables in the solution vector is not an important matter. However, in recent studies of ES and EP, in addition to the decision variables, the solution vector includes a set of strategy parameters specifying the variance of search mutation for each variable and variable combinations. For n decision variables, both methods use an additional number between one and $n(n+1)/2$ such strategy parameters, depending on the degree of freedom the user wants to provide for the search algorithm. These adaptive parameters control the search of each variable, considering its own allowable variance and covariance with other decision variables. We discuss these representations in Section C1.3.2. C1.3.2

In permutation problems, the solutions are usually a vector of node identifiers representing a permutation. Depending on the problem specification, special care is taken in creating valid solutions representing a valid permutation. In these problems, the absolute positioning of the node identifiers is not

as important as the relative positioning of the node identifiers. The representation of permutation problems is discussed further in Section C1.4. C1.4

In early EP works, finite-state machines were used to evolve intelligent algorithms which were operated on a sequence of symbols so as to produce an output symbol which would maximize the algorithm's performance. Finite-state representations were used as solutions to the underlying problem. The input and output symbols were taken from two different finite-state alphabet sets. A solution is represented by specifying both input and output symbols to each link connecting the finite states. The finite-state machine tranforms a sequence of input symbols to a sequence of output symbols. The finite-state representations are discussed in Section C1.5. C1.5

In genetic programming studies, a solution is usually a LISP program specifying a strategy or an algorithm for solving a particular task. Functions and terminals are used to create a vaild solution. The syntax and structure of each function are maintained. Thus, if an OR function is used in the solution, at least two arguments are assigned from the terminal set to make a valid OR operation. Usually, the depth of nestings used in any solution is restricted to a specified upper limit. In recent applications of genetic programming, many special features are used in representing a solution. As the iterations progress, a part of the solution is frozen and defined as a metafunction with specified arguments. We shall discuss these features further in Section C1.6. C1.6

As mentioned earlier, the representation of a solution is important in the working of a search algorithm, including evolutionary algorithms. In EC studies, although a solution can be represented in a number of ways, the efficacy of a representation scheme cannot be judged alone; instead it depends largely on the chosen recombination operators. In the context of *schema processing* and the building block hypothesis, B2.5 it can be argued that a representation that allows good yet important combinations of decision variables to propagate by the action of the search operators is likely to perform well. Radcliffe (1993) outlines a number of properties that a recombination operator must have in order to properly propagate good building blocks. Kargupta *et al* (1992) have shown that the success of GAs in solving a permutation problem coded by three different representations strongly depends on the appropriate recombination operator used. Thus, the choice of a representation scheme must not be made alone, but must be made in conjuction with the choice of the search operators. Guidelines for a suitable representation of decision variables are discussed in Section C1.7. C1.7

C1.1.3 Combined representations

In many search and optimization problems, the solution vector may contain different types of variable. For example, in a mixed-integer programming problem (common to many engineering and decision-making problems) some of the decision variables could be real valued and some could be integer valued. In an engineering gear design problem, the number of teeth in a gear and the thickness of the gear could be two important design variables. The former variable is an integer variable and the latter is a real-valued variable. If the integer variable is coded in five-bit binary strings and the real variable is coded in real numbers, a typical mixed-string representation of the above gear design problem may look like (`10011 23.5`), representing 19 gear teeth and a thickness of 23.5 mm. Sometimes, the variables could be of different types. In a typical civil engineering truss structure problem, the topology of the truss (the connectivity of the truss members represented as presence or absence of members) and the member cross-sectional areas (real valued) are usually the design decision variables. These combined problems are difficult to solve using traditional methods, simply because the search rule in those algorithms does not allow mixed representations. Although there exists a number of mixed-integer programming algorithms such as the *branch-and-bound* method or the *penalty function* method, these algorithms treat the discrete G9.7.4, C5.2 variables as real valued and impose an artificial pressure for these solutions to move towards the desired discrete values. This is achieved either by adding a set of additional constraints or by penalizing infeasible solutions. These algorithms, in general, require extensive computations. However, the string representation of variables in GAs and the flexibility of using a discrete probability distribution for creating solutions in ES and EP studies allow them to be conveniently used to solve such combined problems. In these problems, a solution vector can be formed by concatenating substrings or numerical values representing or specifying each type of variable, as depicted in the above gear design problem representation. Each part of the solution vector is then operated by a suitable recombination and mutation operator. In the above gear design problem representation using GAs, a binary crossover operator may be used for the integer variable represented by the binary string and a real-coded crossover can be used for the continuous variable. Thus,

the recombination operator applied to these mixed representations becomes a collection of a number of operators suitable for each type of variable (Deb 1997). Similar mixed schemes for mutation operators need also to be used for such combined representations. These representations are discussed further in Section C1.8. C1.8

References

Deb K 1995 *Optimization for Engineering Design: Algorithms and Examples* (New Delhi: Prentice-Hall)

——1997 A robust optimal design technique for mechanical component design *Evolutionary Algorithms in Engineering Applications* ed D Dasgupta and Z Michalewicz (Berlin: Springer) in press

Deb K and Agrawal R 1995 Simulated binary crossover for continuous search space *Complex Syst.* **9** 115–48

Chaturvedi D, Deb K and Chakrabarty S K 1995 Structural optimization using real-coded genetic algorithms *Proc. Symp. on Genetic Algorithms (Dehradun)* ed P K Roy and S D Mehta (Dehradun: Mahendra Pal Singh) pp 73–82

Eshelman L J and Schaffer J D 1993 Real-coded genetic algorithms and interval schemata *Foundations of Genetic Algorithms II (Vail, CO)* ed D Whitley (San Mateo, CA: Morgan Kaufmann) pp 187–202

Kargupta H, Deb K and Goldberg D E 1992 Ordering genetic algorithms and deception *Parallel Problem Solving from Nature II (Brussels)* ed R Manner and B Manderick (Amsterdam: North-Holland) pp 47–56

Radcliffe N J 1993 Genetic set recombination *Foundations of Genetic Algorithms II (Vail, CO)* ed D Whitley (San Mateo, CA: Morgan Kaufmann) pp 203–19

Reklaitis G V, Ravindran A and Ragsdell K M 1983 *Engineering Optimization: Methods and Applications* (New York: Wiley)

Schaffer J D, Caruana R A, Eshelman L J and Das R 1989 A study of control parameters affecting online performance of genetic algorithms *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, WA, 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 51–60

Wright A 1991 Genetic algorithms for real parameter optimization *Foundations of Genetic Algorithms (Bloomington, IN)* ed G J E Rawlins (San Mateo, CA: Morgan Kaufmann) pp 205–20

C1.2 Binary strings

Thomas Bäck

Abstract

Binary vectors of fixed length, the standard representation of solutions within canonical genetic algorithms, are discussed in this section, with some emphasis on the question of whether this representation should also be used for problems where the search space fundamentally differs from the space of binary vectors. Focusing on the example of continuous parameter optimization, the schema maximization argument (principle of minimal alphabets), as well as the problem solving argument that the overall optimization problem becomes more complex by introducing a binary representation, is briefly discussed. While a theory on the desirable properties of the decoding function is still lacking, practical experience favors utilization of the most natural representation of solutions rather than enforcing a binary vector representation for other than pseudo-Boolean problems.

The classical representation used in so-called canonical *genetic algorithms* consists of binary vectors (often called bitstrings or binary strings) of fixed length ℓ; that is, the individual space I is given by $I = \{0, 1\}^\ell$ and individuals $\boldsymbol{a} \in I$ are denoted as binary vectors $\boldsymbol{a} = (a_1, \ldots, a_\ell) \in \{0, 1\}^\ell$ (see the book by Goldberg (1989)). The *mutation operator* then typically manipulates these vectors by randomly inverting single variables a_i with small probability, and the *crossover operator* exchanges segments between two vectors to form offspring vectors. B1.2, C3.2.1, C3.3.1

This representation is often well suited to problems where potential solutions have a canonical binary representation, i.e. to so-called pseudo-Boolean optimization problems of the form $f : \{0, 1\}^\ell \to \mathbb{R}$. Some examples of such combinatorial optimization problems are the *maximum-independent-set problem* in graphs, the *set covering problem*, and the *knapsack problem*, which can be represented by binary vectors simply by including (excluding) a vertex, set, or item i in (from) a candidate solution when the corresponding entry $a_i = 1$ ($a_i = 0$). G9.2, G9.6, G9.7

Canonical genetic algorithms, however, also emphasize the binary representation in the case of problems $f : S \to \mathbb{R}$ where the search space S fundamentally differs from the binary vector space $\{0, 1\}^\ell$. The most prominent example of this is given by the application of canonical genetic algorithms for continuous parameter optimization problems $f : \mathbb{R}^n \to \mathbb{R}$ as outlined by Holland (1975) and empirically investigated by De Jong (1975). The mechanisms of encoding and decoding between the two different spaces $\{0, 1\}^\ell$ and $\mathbb{R}^n$ then require us to restrict the continuous space to finite intervals $[u_i, v_i]$ for each variable $x_i \in \mathbb{R}$, to divide the binary vector into n segments of (in most cases) equal length ℓ_x, such that $\ell = n\ell_x$, and to interpret a subsegment $(a_{(i-1)\ell_x+1}, \ldots, a_{i\ell_x})$ $(i = 1, \ldots, n)$ as the binary encoding of the variable x_i. Decoding then either proceeds according to the standard binary decoding function $\Gamma^i : \{0, 1\}^\ell \to [u_i, v_i]$, where (see Bäck 1996)

$$\Gamma^i(a_1, \ldots, a_\ell) = u_i + \frac{v_i - u_i}{2^{\ell_x} - 1} \left(\sum_{j=0}^{\ell_x - 1} a_{i\,\ell_x - j}\, 2^j \right) \tag{C1.2.1}$$

or by using a Gray code interpretation of the binary vectors, which ensures that adjacent integer values are represented by binary vectors with Hamming distance one (i.e. they differ by one entry only). For the

Gray code, equation (C1.2.1) is extended by a conversion of the Gray code representation to the standard code, which can be done for example according to

$$\Gamma^i(a_1,\ldots,a_{\ell_x}) = u_i + \frac{v_i - u_i}{2^{\ell_x}-1}\left(\sum_{j=0}^{\ell_x-1}\left(\bigoplus_{k=1}^{\ell_x-j} a_{(i-1)\ell_x+k}\right)2^j\right) \tag{C1.2.2}$$

where $\oplus$ denotes addition modulo two.

It is clear that this mapping from the representation space $I = \{0,1\}^\ell$ to the search space $S = \prod_{i=1}^n [u_i, v_i]$ is injective but not surjective, i.e. not all points of the search space are represented by binary vectors, such that the genetic algorithm performs a grid search and, depending on the granularity of the grid, might fail to locate an optimum precisely (notice that ℓ_x and the range $[u_i, v_i]$ determine the distance of two grid points in dimension i according to $\Delta x_i = (v_i - u_i)/(2^{\ell_x} - 1)$). Moreover, both decoding mappings given by equations (C1.2.1) and (C1.2.2) introduce additional nonlinearities to the overall objective function $f' : \{0,1\}^\ell \to \mathbb{R}$, where $f'(\boldsymbol{a}) = (f \circ \times_{i=1}^n \Gamma^i)(\boldsymbol{a})$, and the standard code according to equation (C1.2.1) might cause the problem f' to become harder than the original optimization problem f (see the work of Bäck (1993, 1996, ch 6) for a more detailed discussion).

While parameter optimization is still the dominant field where canonical genetic algorithms are applied to problems in which the search space is fundamentally different from the binary space $\{0,1\}^\ell$, there are other examples as well, such as the traveling salesman problem (Bean 1993) and job shop scheduling (Nakano and Yamada 1991). Here, rather complex mappings from $\{0,1\}^\ell$ to the search space were defined—to improve their results, Yamada and Nakano (1992) later switched to a more canonical integer representation space, giving a further indication that the problem characteristics should determine the representation and not vice versa.

The reasons why a binary representation of individuals in genetic algorithms is favored by some researchers can probably be split into historical and *schema-theoretical* aspects. Concerning the history, B2.5
it is important to notice that Holland (1975, p 21) does not define adaptive plans to work on binary variables (alleles) $a_i \in \{0,1\}$, but allows arbitrary but finite individual spaces $I = A_1 \times \ldots \times A_\ell$, where $A_i = \{\alpha_{i_1}, \ldots, \alpha_{i_{k_i}}\}$. Furthermore, his notion of *schemata* (certain subsets of I characterized by the fact that all members—so-called instances of a schema—share some similarities) does not require binary variables either, but is based on extending the sets A_i defined above by an additional 'don't care' symbol (Holland 1975, p 68). For the application example of parameter optimization, however, he chooses a binary representation (Holland 1975, pp 57, 70), probably because this is the canonical way to map the continuous object variables to the discrete allele sets A_i defined in his adaptive plans, which in turn are likely to be discrete because they aim at modeling the adaptive capabilities of natural evolution on the genotype level.

Interpreting a genetic algorithm as an algorithm that processes schemata, Holland (1975, p 71) then argues that the number of schemata available under a certain representation is maximized by using binary variables; that is, the maximum number of schemata is processed by the algorithm if $a_i \in \{0,1\}$. This result can be derived by noticing that, when the cardinality of an alphabet A for the allele values is $k = |A|$, the number of different schemata is $(k+1)^\ell$ (i.e. 3^ℓ in the case of binary variables). For binary alleles, 2^ℓ different solutions can be represented by vectors of length ℓ, and in order to encode the same number of solutions by a k-ary alphabet, a vector of length

$$\ell' = \ell\frac{\ln 2}{\ln k} \tag{C1.2.3}$$

is needed. Such a vector, however, is an instance of $(k+1)^{\ell'}$ schemata, a number that is always smaller than 3^ℓ for $k > 2$; that is, fewer schemata exist for an alphabet of higher cardinality, if the same number of solutions is represented.

Goldberg (1989, p 80) weakens the general requirement for a binary alphabet by proposing the so-called *principle of minimal alphabets*, which states that 'The user should select the smallest alphabet that permits a natural expression of the problem' (presupposing, however, that the binary alphabet permits a natural expression of continuous parameter optimization problems and is no worse than a real-valued representation (Goldberg 1991)). Interpreting this strictly, the requirement for binary alphabets can be dropped, as many practitioners (e.g. Davis 1991 and Michalewicz 1996) who apply (noncanonical) genetic algorithms to industrial problems have already done, using nonbinary, problem-adequate representations

such as *real-valued vectors*, integer lists representing *permutations*, *finite-state machine representations*, and *parse trees.* C1.3, C1.4, C1.5 C1.6

At present, there are neither clear theoretical nor empirical arguments that a binary representation should be used for arbitrary problems other than those that have a canonical representation as pseudo-Boolean optimization problems. From an optimization point of view, where the quality of solutions represented by individuals in a population is to be maximized, the interpretation of genetic algorithms as schema processors and the corresponding *implicit parallelism* and *schema theorem* results are of no practical use. B2.5 From our point of view, the decoding function $\Gamma : \{0,1\}^\ell \rightarrow S$ that maps the binary representation to the canonical search space a problem is defined on plays a much more crucial role than the schema processing aspect, because, depending on the properties of Γ, the overall pseudo-Boolean optimization problem $f' = f \circ \Gamma$ might become more complex than the original search problem $f : S \rightarrow \mathbb{R}$. Consequently, one might propose the requirement that, if a mapping between representation space and search space is used at all, it should be kept as simple as possible and obey some structure preserving conditions that still need to be formulated as a guideline for finding a suitable encoding.

References

Bäck T 1993 Optimal mutation rates in genetic search *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 2–8

——1996 *Evolutionary Algorithms in Theory and Practice* (New York: Oxford University Press)

Bean J C 1993 *Genetics and Random Keys for Sequences and Optimization* Technical Report 92–43, University of Michigan Department of Industrial and Operations Engineering

Davis L 1991 *Handbook of Genetic Algorithms* (New York: Van Nostrand Reinhold)

De Jong K A 1975 *An Analysis of the Behaviour of a Class of Genetic Adaptive Systems* PhD Thesis, University of Michigan

Goldberg D E 1989 *Genetic Algorithms in Search, Optimization and Machine Learning* (Reading, MA: Addison Wesley)

——1991 The theory of virtual alphabets *Parallel Problem Solving from Nature—Proc. 1st Workshop, PPSN I (Lecture Notes in Computer Science 496)* ed H-P Schwefel and R Männer (Berlin: Springer) pp 13–22

Holland J H 1975 *Adaptation in Natural and Artificial Systems* (Ann Arbor, MI: University of Michigan Press)

Michalewicz Z 1996 *Genetic Algorithms + Data Structures = Evolution Programs* (Berlin: Springer)

Nakano R and T Yamada 1991 Conventional genetic algorithms for job shop problems *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, July 1991)* ed R K Belew and L B Booker (San Mateo, CA: Morgan Kaufmann) pp 474–9

Yamada T and R Nakano 1992 A genetic algorithm applicable to large-scale job-shop problems *Parallel Problem Solving from Nature, 2 (Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature, Brussels, 1992)* ed R Männer and B Manderick (Amsterdam: Elsevier) pp 281–90

C1.3 Real-valued vectors

David B Fogel

Abstract

Real-valued vector representations are described and consideration is given to the relationship between the cardinality of the alphabet of symbols and implicit parallelism. Attention is also given to mechanisms of altering data structures under real-valued representation.

C1.3.1 Object variables

When posed with a real-valued function optimization problem of the form find the vector $\boldsymbol{x}$ such that $F(\boldsymbol{x}) : \mathbb{R}^n \rightarrow \mathbb{R}$ is minimized (or maximized), *evolution strategies* (Bäck and Schwefel 1993) and *evolutionary programming* (Fogel 1995, pp 75–84, 136–7) typically operate directly on the real-valued vector $\boldsymbol{x}$ (with the components of $\boldsymbol{x}$ identified as *object parameters*). In contrast, traditional *genetic algorithms* operate on a coding (often binary) of the vector $\boldsymbol{x}$ (Goldberg 1989, pp 80–4). The choice to use a separate coding rather than operating on the parameters themselves relies on the fundamental belief that it is useful to operate on subsections of a problem and try to optimize these subsections (i.e. building blocks) in isolation, and then subsequently recombine them so as to generate improved solutions. More specifically, Goldberg (1989, p 80) recommends

> The user should select a coding so that short, low-order schemata are relevant to the underlying problem and relatively unrelated to schemata over other fixed positions.
>
> The user should select the smallest alphabet that permits a natural expression of the problem.

Although the smallest alphabet generates the greatest implicit parallelism (see Section B2.5), there is no empirical evidence to indicate that binary codings allow for greater effectiveness or efficiency in solving real-valued optimization problems (see the tutorial by Davis (1991, p 63) for a commentary on the ineffectiveness of binary codings).

Evolution strategies and evolutionary programming are not generally concerned with the recombination of building blocks in a solution and do not consider schema processing. Instead, solutions are viewed in their entirety, and no attempt is made to decompose whole solutions into subsections and assign credit to these subsections.

With the belief that maximizing the number of schemata being processed is not necessarily useful, or may even be harmful (Fogel and Stayton 1994), there is no compelling reason in a real-valued optimization problem to act on anything except the real values of the vector $\boldsymbol{x}$ themselves. Moreover, there has been a general trend away from binary codings within genetic algorithm research (see e.g. Davis 1991, Belew and Booker 1991, Forrest 1993, and others). Michalewicz (1992, p 82) indicated that for real-valued numerical optimization problems, floating-point representations outperform binary representations because they are more consistent and more precise and lead to faster execution. This trend may reflect a growing rejection of the building block hypothesis as an explanation for how genetic algorithms act as optimization procedures (see Section B2.5).

With evolution strategies and evolutionary programming, the typical method for searching a real-valued solution space is to add a multivariate zero-mean Gaussian random variable to each parent involved in the creation of offspring (see Section C3.2.2). In consequence, this necessitates the setting of the

covariance matrix for the Gaussian perturbation. If the covariances between parameters are ignored, only a vector of standard deviations in each dimension is required. There are a variety of methods for setting these standard deviations. Section C3.2.2 offers a variety of procedures for mutating real-valued vectors. C3.2.2

C1.3.2 Object variables and strategy parameters

It has been recognized since 1967 (Rechenberg 1994, Reed *et al* 1967) that it is possible for each solution to possess an auxiliary vector of parameters that determine how the solution will be changed. Two general procedures for adjusting the object parameters via Gaussian mutation have been proposed (Schwefel 1981, Fogel *et al* 1991) (see Section C3.2.2). In each case, a vector of *strategy parameters* for self-adaptation is included with each solution and is subject to random variation. The vector may include covariance or rotation information to indicate how mutations in each parameter may covary. Thus the representation consists of two or three vectors:

$$(\boldsymbol{x}, \boldsymbol{\sigma}, \boldsymbol{\alpha})$$

where $\boldsymbol{x}$ is the vector of object parameters $(x_1, \ldots, x_n)$, $\boldsymbol{\sigma}$ is the vector of standard deviations, and $\boldsymbol{\alpha}$ is the vector of rotation angles corresponding to the covariances between mutations in each dimension, and may be omitted if these covariances are set to be zero. The vector $\boldsymbol{\sigma}$ may have n components $(\sigma_1, \ldots, \sigma_n)$ where each entry corresponds to the standard deviation in the ith dimension, $i = 1, \ldots, n$. The vector $\boldsymbol{\sigma}$ may also degenerate to a scalar σ in which case this single value is used as the standard deviation in all dimensions. Intermediate numbers of standard deviations are also possible, although such implementation is uncommon (this also applies to the rotation angles α). Very recent efforts by Ostermeier *et al* (1994) offer a variation on the methods of Schwefel (1981) and further study is required to determine the general effectiveness of this new technique (see Section C3.2.2).

Recent efforts in genetic algorithms have also included *self-adaptive procedures* (see e.g. Spears 1995) C7.1 and these may incorporate similar real-valued coding for variation operators including crossover and point mutations, on both real-valued or binary or otherwise coded object parameters.

References

Bäck T and Schwefel H-P 1993 An overview of evolutionary algorithms for parameter optimization *Evolutionary Comput.* **1** 1–24

Belew R K and Booker L B (eds) 1991 *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, July 1991)* (San Mateo, CA: Morgan Kaufmann)

Davis L 1991 A genetic algorithms tutorial IV. Hybrid genetic algorithms *Handbook of Genetic Algorithms* ed L Davis (New York: Van Nostrand Reinhold)

Fogel D B 1995 *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence* (Piscataway, NJ: IEEE)

Fogel D B, Fogel L J and Atmar J W 1991 Meta-evolutionary programming *Proc. 25th Asilomar Conf. on Signals, Systems, and Computers* ed R R Chen (San Jose, CA: Maple) pp 540–5

Fogel D B and Stayton L C 1994 On the effectiveness of crossover in simulated evolutionary optimization *BioSystems* **32** 171–82

Forrest S (ed) 1993 *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* (San Mateo, CA: Morgan Kaufmann)

Goldberg D E 1989 *Genetic Algorithms in Search, Optimization and Machine Learning* (Reading, MA: Addison-Wesley)

Michalewicz Z 1992 *Genetic Algorithms + Data Structures = Evolution Programs* (Berlin: Springer)

Ostermeier A, Gawelczyk A and Hansen N 1994 A derandomized approach to self-adaptation of evolution strategies *Evolutionary Comput.* **2** 369–80

Rechenberg I 1994 Personal communication

Reed J, Toombs R and Barricelli N A 1967 Simulation of biological evolution and machine learning *J. Theor. Biol* **17** 319–42

Schwefel H-P 1981 *Numerical Optimization of Computer Models* (Chichester: Wiley)

Spears W M 1995 Adapting crossover in evolutionary algorithms *Evolutionary Programming IV: Proc. 4th Ann. Conf. on Evolutionary Programming (San Diego, CA, March 1995)* ed J R McDonnell, R G Reynolds and D B Fogel (Cambridge, MA: MIT Press) pp 367–84

C1.4 Permutations

Darrell Whitley

Abstract

There is a well developed body of literature in computer science describing properties of permutations and algorithms for manipulating permutations. Permutations have also been a popular representation for some classic combinatorial optimization problems such as the traveling salesman problem. Some basic properties of permutations are reviewed, as well as how infinite population models of genetic algorithms and random heuristic search can be adapted for problems with a permutation-based representation. The notion of schemata for permutation-based representations is also examined.

C1.4.1 Introduction

To quote Knuth (1973), 'A permutation of a finite set is an arrangement of its elements into a row.' Given n unique objects, $n!$ permutations of the objects exist. There are various properties of permutations that are relevant to the manipulation of permutation representations by evolutionary algorithms, both from a representation point of view and from an analytical perspective.

As researchers began to apply evolutionary algorithms to applications that are naturally represented as permutations, it became clear that these problems pose different coding challenges than traditional parameter optimization problems. First, for some types of problem there are multiple equivalent solutions. When a permutation is used to represent a cycle, as in the *traveling salesman problem* (TSP), then all shifts G9.5 of the permutation are equivalent solutions. Furthermore, all reversals of a permutation are also equivalent solutions. Such symmetries can pose problems for evolutionary algorithms that rely on recombination.

Another problem is that permutation problems cannot be processed using the same general recombination and mutation operators which are applied to parameter optimization problems. The use of a permutation representation may in fact mask very real differences in the underlying combinatorial optimization problems. An example of these differences is evident in the description of classic problems such as the TSP and the problem of *resource scheduling*. F1.5

The traveling salesman problem is the problem of visiting each vertex (i.e. city) in a full connected graph exactly once while minimizing a cost function defined with respect to the edges between adjacent vertices. In simple terms, the problem is to minimize the total distance traveled while visiting all the cities and returning to the point of origin. The TSP is closely related to the problem of finding a Hamiltonian circuit in an arbitrary graph. The Hamiltonian circuit is a set of edges that form a cycle which visits every vertex exactly once. It is relatively easy to show that the problem of finding a set of Boolean values that yield an evaluation of 'true' for a three-conjunction normal form Boolean expression is directly *polynomial-time reducible* to the problem of finding a Hamiltonian circuit in a specific type of graph (Cormen *et al* 1990). The Hamiltonian circuit problem in turn is reducible to the TSP. All of these problems have a nondeterministic polynomial-time (NP) solution but have no known polynomial-time solution. These problems are also members of the set of hardest problems in NP, and hence are NP complete.

Permutations are also important for scheduling applications, variants of which are also often NP complete. Some scheduling problems are directly related to the TSP. Consider minimizing setup times between a set of N jobs, where the function Setup(A, B) is the cost of switching from job A to job B. If Setup(A, B) = Setup(B, A) this is a variant of the symmetric TSP, except that the solution maybe a

path instead of a cycle through the graph (i.e. it visits every vertex, but does not necessarily return to the origin.) The TSP and setup minimization problem may also be nonsymmetric: the cost of going from vertex A to B may not be equal to the cost of going from vertex B to A.

Other types of scheduling problem are different from the TSP. Assume that one must schedule service times for a set of customers. If this involves access to a critical resource, then those customers that are scheduled early may have access to resources that are unavailable to later customers. If one is scheduling appointments, for example, later customers will have less choice with respect to which time slots are available to them. In either case, access to limited resources is critical. We would like to optimize the match between resources and customers. This could allow us to give more customers what they want in terms of resources, or the goal might be to increase the number of customers who can be serviced. In either case, permutations over the set of customers can be used as a priority queue for scheduling. While there are various classic problems in the scheduling literature, the term *resource scheduling* is used here to refer to scheduling applications where resources are consumed.

Permutations are also sometimes used to represent multisets. A multiset is also sometimes referred to as a *bag*, which is analogous to a set except that a bag may contain multiple copies of identical elements. In sets, the duplication of elements is not significant, so that

$$\{a, a, b\} = \{a, b\}.$$

However, in the following multiset,

```
M = {a, a, b, b, b, c, d, e, e, f, f}
```

there are two `a`'s, three `b`'s, one `c`, one `d`, two `e`'s and two `f`'s, and duplicates are significant. In scheduling applications that map jobs to machines, it may be necessary to schedule two jobs of type `a`, three jobs of type `b`, and so on. Note that it is not necessary that all jobs of type `a` be scheduled contiguously. While M in the above illustration contains 11 elements, there are not 11! unique permutations. Rather, the number of unique permutations is given by

$$\frac{11!}{2!3!1!1!2!2!}$$

and in general

$$\frac{n!}{n_1!n_2!n_3!\ldots}$$

where n is the number of elements in the multiset and n_i is the number of elements of type i (Knuth 1973). Radcliffe (1993) considers the application of genetic and evolutionary operators when the solution is expressed as a set or multiset (bag).

Before looking in more detail at the relationship between permutations and evolutionary algorithms, some general properties of permutations are reviewed that are both interesting and useful.

C1.4.2 Mapping integers to permutations

The set of $n!$ permutations can be mapped onto the set of integers in various ways. Whitley and Yoo (1995) give the following algorithm which converts an integer X into the corresponding permutation.

(i) Choose some ordering of the permutation which is defined to be sorted.
(ii) Sort and index the N elements ($N \geq 1$) of the permutation from 1 to N. Pick an index X for a specific permutation such that $0 \leq X < N!$.
(iii) If $X = 0$, pick all remaining elements in the sorted permutation list in the sequence in which they occur and stop.
(iv) IF $X < (N-1)!$ pick the first element of the remaining list; GOTO (vi). Otherwise, continue.
(v) Find Y such that $(Y-1)(N-1)! \leq X < Y(N-1)!$. The Yth element of the sort list is the next element of the permutation. $X = X - (Y-1)((N-1)!)$.
(vi) Delete the chosen element from the list of sorted elements; $N = N - 1$; GOTO (iii).

This algorithm can also be inverted to map integers to permutations. For permutations of length three this generates the following correspondance:

```
X = 0 indexes 123      X = 3 indexes 231
X = 1 indexes 132      X = 4 indexes 312
X = 2 indexes 213      X = 5 indexes 321.
```

C1.4.3 The inverse of a permutation

One important property of a permutation is that it has a well-defined *inverse* (Knuth 1973). Let $a_1a_2 \ldots a_n$ be a permutation of the set $\{1, 2, \ldots, n\}$. This can be written in a two-line form

$$\begin{pmatrix} 1 & 2 & 3 & \ldots & n \\ a_1 & a_2 & a_3 & \ldots & a_n \end{pmatrix}.$$

The inverse is obtained by reordering both rows such that the second row is transformed into the sequence $123 \ldots n$; the reordering of the first that occurs as a consequence of reordering the second row yields the inverse of permutation $a_1a_2a_3 \ldots a_n$. The inverse is denoted $a'_1a'_2a'_3 \ldots a'_n$. Knuth (1973) gives the following example of a permutation, 5 9 1 8 2 6 4 7 3, and shows that its inverse can be obtained as follows:

$$\begin{pmatrix} 591826473 \\ 123456789 \end{pmatrix} = \begin{pmatrix} 123456789 \\ 359716842 \end{pmatrix}$$

which yields the inverse 3 5 9 7 1 6 8 4 2. Knuth also points out that $a'_j = k$ if and only if $a_k = j$. The inverse can be used as part of a function for mapping permutations to a canonical form, which in turn makes it easier to model problems with permutation representations.

C1.4.4 The mapping function

When modeling evolutionary algorithms it is often useful to compute a transmission function $r_{i,j}(k)$ which yields the probability of recombining strings i and j and obtaining an arbitrary string k. Whitley and Yoo (1995) explain how to compute the transmission function for a single string k and then to generalize the results to all other strings. In this case, the strings represent permutations and the remapping function, denoted @, functions as follows:

$$r_{3421,1342}(3124) = r_{3421@3124,1342@3124}(1234).$$

The computation $Y = A@X$ behaves as follows. Let any permutation X be represented by $x_1x_2x_3 \ldots x_n$. Then $a_1a_2a_3 \ldots a_n@x_1x_2x_3 \ldots x_n$ yields $Y = y_1y_2y_3 \ldots y_n$ where $y_i = j$ when $a_i = x_j$. Thus, (3421@3124) yields (1432) since $(a_1 = 3 = x_1) \Rightarrow (y_1 = 1)$. Next, $(a_2 = 4 = x_4) \Rightarrow (y_2 = 4)$, $(a_3 = 2 = x_3) \Rightarrow (y_3 = 3)$, and $(a_4 = 1 = x_2) \Rightarrow (y_4 = 2)$. This mapping function is analogous to the bitwise addition (mod 2) used to reorder the vector s for binary strings. However, note that $A@X \neq X@A$. Furthermore, for permutation recombination operators it is *not generally true* that $r_{i,j} = r_{j,i}$.

This allows one to compute the transmission function with respect to a canonical permutation, in this case 1234, and generalize this mapping to all other permutations. This mapping can be achieved by simple element substitution. First, the function r can be generalized as follows:

$$r_{3421,1342}(3124) = r_{wzyx,xwzy}(wxyz)$$

where w, x, y, and z are variables representing the elements of the permutation (e.g. $w = 3, x = 1, y = 2, z = 4$). If $wxyz$ now represents the canonically ordered permutation 1234,

$$r_{wzyx,xwzy}(wxyz) = r_{1432,2143}(1234) = r_{3421@3124,1342@3124}(1234).$$

We can also relate this mapping operator to the process of finding an inverse. The permutations in the expression

$$r_{3421,1342}(3124) = r_{1432,2143}(1234)$$

are included as rows in an array. To map the left-hand side of the preceding expression to the terms in the right-hand side, first compute the inverses for each of the terms in the left-hand side:

$$\begin{pmatrix} 3421 \\ 1234 \end{pmatrix} = \begin{pmatrix} 1234 \\ 4312 \end{pmatrix}$$
$$\begin{pmatrix} 1342 \\ 1234 \end{pmatrix} = \begin{pmatrix} 1234 \\ 1423 \end{pmatrix}$$
$$\begin{pmatrix} 3124 \\ 1234 \end{pmatrix} = \begin{pmatrix} 1234 \\ 2314 \end{pmatrix}$$

Collect the three inverses into a single array. We also then add 1 2 3 4 to the array and inverse the permutation 2 3 1 4, at the same time rearranging all the other permutations in the array:

$$\begin{pmatrix} 4312 \\ 1423 \\ 2314 \\ 1234 \end{pmatrix} = \begin{pmatrix} 1432 \\ 2143 \\ 1234 \\ 3124 \end{pmatrix}.$$

This yields the permutations 1432, 2143, and 1234 which represent the desired canonical form as it relates to the notion of substitution into a symbolic canonical form. One can also reverse the process to find the permutations p_i and p_j in the following context:

$$r_{1432,2143}(1234) = r_{p_i,p_j}(3124) = r_{p_i@3124,\,p_j@3124}(1234).$$

C1.4.5 Matrix representations

When comparing the TSP to the problem of resource scheduling, in one case adjacency is important (the TSP) and in the other case relative order is important (resource scheduling). One might also imagine problems where absolute position is important. One way in which the differences between adjacency and relative order can be illustrated is to use a matrix representation of the information contained in a permutation.

When we discuss adjacency in the TSP, we typically are referring to a symmetric TSP: the distance of going from city A to B is the same as going from B to A. Thus, when we define a matrix representing a tour, there will be two edges in every row of the matrix, where a row–column entry of 1 represents an edge connecting the two cities. Thus, the matrices for the tours [A B C D E F] and [C D E B F A] (left and right, respectively) are as follows:

	A	B	C	D	E	F
A	0	1	0	0	0	1
B	1	0	1	0	0	0
C	0	1	0	1	0	0
D	0	0	1	0	1	0
E	0	0	0	1	0	1
F	1	0	0	0	1	0

	A	B	C	D	E	F
A	0	0	1	0	0	1
B	0	0	0	0	1	1
C	1	0	0	1	0	0
D	0	0	1	0	1	0
E	0	1	0	1	0	0
F	1	1	0	0	0	0

One thing that is convenient about the matrix representation is that it is easy to extract information about where common edges occur. This can also be expressed in the form of a matrix, where a zero or one respectively is placed in the matrix where there is agreement in the two parent structures. If the values in the parent matrices conflict, we will place a # in the matrix. Using the two above structures as parents, the following common information is obtained:

	A	B	C	D	E	F
A	0	#	#	0	0	1
B	#	0	#	0	#	#
C	#	#	0	1	0	0
D	0	0	1	0	1	0
E	0	#	0	1	0	#
F	1	#	0	0	#	0

This matrix can be interpreted in the following way. If we convert the # symbols to * symbols, then (in the notation typically used by the genetic algorithm community) a hyperplane is defined in this binary space in which both of the parents reside. If a recombination operator is applied, the offspring should also reside in this same subspace (this is the concept of *respect*, as used by Radcliffe (1991) ; note mutation can still be applied after recombination).

This matrix representation does bring out one feature rather well: the common subtour information can automatically and easily be extracted and passed on to the offspring during recombination.

The matrix defining the common hyperplane information also defines those offspring that represent a recombination of the information contained in the parent structures. In fact, any assignment of 0 or 1 bits to the locations occupied by the # symbols could be considered valid recombinations, *but not all are feasible solutions to the TSP*, because not all recombinations result in a Hamiltonian circuit. We would like to have an offspring that is not only a valid recombination, but also a feasible solution.

The matrix representation can also make explicit relative order information. Consider the same two parents: [A B C D E F] and [C D E B F A]. Relative order can be represented as follows. Each row will be the relative order information for a particular element of a permutation. The columns will be all permutation elements in some canonical order. If A is the first element in a permutation, then a one will be placed in every column (except column A; the diagonal will again be zero) to indicate A precedes all other cities. This representation is given by Fox and McMahon (1991). Thus, the matrices for [A B C D E F] and [C D E B F A] are as follows (left and right, respectively):

	A	B	C	D	E	F
A	0	1	1	1	1	1
B	0	0	1	1	1	1
C	0	0	0	1	1	1
D	0	0	0	0	1	1
E	0	0	0	0	0	1
F	0	0	0	0	0	0

	A	B	C	D	E	F
A	0	0	0	0	0	0
B	1	0	0	0	0	1
C	1	1	0	1	1	1
D	1	1	0	0	1	1
E	1	1	0	0	0	1
F	1	0	0	0	0	0

In this case, the lower triangle of the matrix flags *inversions*, which should not be confused with an *inverse*. If $a_1 a_2 a_3 \ldots a_n$ is a permutation of the canonically ordered set $1, 2, 3, \ldots, n$ then the pair (a_i, a_j) is an *inversion* if $i < j$ and $a_i > a_j$ (Knuth 1973). Thus, the number of 1 bits in the lower triangles of the above matrices is also a count of the number of inversions (which should also not be confused with the *inversion* operator used in simple genetic algorithms, see Holland 1975, p 106, Goldberg 1989, p 166).

The common information can also extracted as before. This produces the following matrix:

	A	B	C	D	E	F
A	0	#	#	#	#	#
B	#	0	#	#	#	1
C	#	#	0	1	1	1
D	#	#	0	0	1	1
E	#	#	0	0	0	1
F	#	0	0	0	0	0

Note that this binary matrix is again symmetric around the diagonal, except that the lower triangle and upper triangle have complementary bit values. Thus only $N(N-1)/2$ elements are needed to represent relative order information.

There have been few studies of how recombination crossover operators generate offspring in this particular representation space. Fox and McMahon (1991) offer some work of this kind and also define several operators that work directly on these binary matrices for relative order.

While matrices may not be the most efficient form of implementation, they do provide a tool for better understanding sequence recombination operators designed to exploit relative order. It is clear that adjacency and relative order relationships are different and are best expressed by different binary matrices. Likewise, absolute position information also has a different matrix representation (for example, rows could represent cities and the columns represent positions). *Cycle crossover* (see Starkweather *et al* 1991, Oliver *et al* 1987) appears to be a good absolute position operator, although it is hard to find problems in the literature where absolute position is critical. C3.3.3.6

C1.4.6 Alternative representations

Let $\boldsymbol{P}$ be an arbitrary permutation and P_j be the jth element of the permutation. One notable alternative representation of a permutation is to define some canonical ordering, $\boldsymbol{C}$, over the elements in the permutation and then define a vector of integers, $\boldsymbol{I}$, such that the integer in position j corresponds to the position in which element C_j appears in $\boldsymbol{P}$. Such a vector $\boldsymbol{I}$ can then serve as a representation of a permutation. More precisely,

$$P_{(I_j)} = C_j.$$

To illustrate:

$$\boldsymbol{C} = a\ b\ c\ d\ e\ f\ g\ h$$

$$\boldsymbol{I} = 6\ 2\ 5\ 3\ 8\ 7\ 1\ 4 \quad \text{which represents} \quad \boldsymbol{P} = g\ b\ d\ h\ c\ a\ f\ e.$$

This may seem like a needless indirection, but consider that I can be generalized to allow a larger number of possible values than there are permutation elements. I can also be generalized to allow all real values (although for computer implementations the distinction is somewhat artificial since all digital

representations of real values are discrete and finite). We now have a parameter-based presentation of the permutation such that we can generate random vectors I representing permutations. If the number of values for which elements in I are defined is dramatically larger than the number of elements in the permutation, then duplicate values in randomly generated vectors will occur with very small probability.

This representation allows a permutation problem to be treated as if it were a more traditional parameter optimization problem with the constraint that no two elements of vector I should be equal, or that there is a well defined way to resolve ties. Evolutionary algorithm techniques normally used for parameter optimization problems can thus be applied to permutation problems using this representation.

This idea has been independently invented on a couple of occasions. The first use of this coding method was by Steve Smith of Thinking Machines. A version of this coding was used by the ARGOT Strategy (Shaefer 1987) and the representation was picked up by Syswerda (1989) and by Schaffer *et al* (1989) for the TSP. More recently, a similar idea was introduced by Bean (1994) under the name *random keys*.

C1.4.7 Ordering schemata and other metrics

Goldberg and Lingle (1985) built on earlier work by Franz (1972) to describe similarity subsets between different permutations. Franz's calculations were related to the use of inversion operators for traditional genetic algorithm binary representations. The use of inversion operators is very much relevant to the topic of permutations, since in order to apply inversion the binary alleles must be tagged in some way and inversion acts in the space of all possible permutations of allele orderings. Thus,

```
((6 0)(3 1)(2 0)(8 1)(1 0)(5 1)(7 0)(4 0))
```

is equivalent to

```
((1 0)(2 0)(3 1)(4 0)(5 1)(6 0)(7 0)(8 1))
```

which represents the binary string 00101001 in a position-independent fashion (Holland 1975).

Goldberg and Lingle were more directly concerned with problems where the permutation was itself the problem representation, and, in particular, they present early results for the TSP. They also introduced the partially mapped crossover (PMX) operator and the notion of *ordering schemata*, or *o-schemata*. For o-schemata, the symbol ! acts as a wild card match symbol. Thus, the template

```
!  !  1 !  !  7 3 !
```

represents all permutations with a one as the third element, a seven as the sixth element, and a three as the seventh element. Given o selected positions in a permutation of length l, there are $(l-o)!$ permutations that match an o-schemata. One can also count the number of possible o-schema. There are clearly $\binom{l}{o}$ ways to choose o fixed positions; there are also $\binom{l}{o}$ ways to pick the permutation elements that fill the slots, and $o!$ ways of ordering the elements (i.e. the number of permutations over the chosen combination of subelements). Thus, Goldberg (1989, Goldberg and Lingle 1985) notes that the total number of o-schemata, n_{os}, can be calculated by

$$n_{\mathrm{os}} = \sum_{j=0}^{l} \frac{l!}{(l-j)!\,j!}\frac{l!}{(l-j)!}.$$

Note that in this definition of the o-schemata, relative order is not accounted for. In other words, if relative order is important then all of the following shifted o-schemata,

```
1  !  !  7  3  !  !  !
!  1  !  !  7  3  !  !
!  !  1  !  !  7  3  !
!  !  !  1  !  !  7  3
```

could be viewed as equivalent. Such schemata may or may not 'wrap around'. Goldberg discusses o-schemata which have an absolute fixed position (o-schemata, type a) and those with relative position which are shifts of a specified template (o-schemata, type r).

This work on o-schemata predates the distinctions between relative order permutation problems, absolute position problems, and adjacency-based problems. Thus, o-schemata appear to be better for understanding resource scheduling applications than for the TSP. In subsequent work, Kargupta *et al* (1992) attempt to use ordering schemata to construct *deceptive functions* for ordering problems—that is, problems where the average fitness values of the o-schemata provide misleading information. Note that

B2.7.1

such problems are constructed to mislead simple genetic algorithms and may or may not be difficult with respect to other types of algorithm. (For a discussion of deception see the article by Goldberg (1987) and Whitley (1991) and for another perspective see the article by Grefenstette (1993).) The analysis of Kargupta *et al* (1992) considers PMX, a uniform ordering crossover operator (UOX), and a relative ordering crossover operator (ROX).

An alternative way of constructing relative order problems and of comparing the similarity of permutations is given by Whitley and Yoo (1995). Recall that a relative order matrix that has a 1 bit in position (X, Y) if row element X appears before column element Y in a permutation. Note that the matrix representation yields a unique binary representation for each permutation. Using this representation one can also define the Hamming distance between two permutations P1 and P2; Hamming distance is denoted by HD(index(P1), index(P2)), where the permutations are represented by their integer index. In the following examples, the Hamming distance is computed with respect to the lower triangle (i.e. it is a count of the number of 1 bits in the lower triangle):

```
                A B C D
               ---------
          A  | 0 1 1 1
A B C D   B  | 0 0 1 1          HD(0,0) = 0
          C  | 0 0 0 1
          D  | 0 0 0 0

                A B C D
               ---------
          A  | 0 0 0 0
B D C A   B  | 1 0 1 1          HD(0,11) = 4
          C  | 1 0 0 0
          D  | 1 0 1 0


                A B C D
               ---------
          A  | 0 0 0 0
D C B A   B  | 1 0 0 0         HD(0,23) = 6
          C  | 1 1 0 0
          D  | 1 1 1 0
```

Whitley and Yoo (1995) point out that this representation is not perfect. Since $2^{(N^2)} > N!$, certain binary strings are undefined. For example, consider the following upper triangle:

```
1 1 1
  0 1
    0
```

Element 1 occurs before 2, 3, and 4, which poses no problem, but 2 occurs after 3, 2 occurs before 4, and 4 occurs before 3. Using $>$ to denote relative order, this implies a nonexistent ordering such that

```
3 > 2 > 4   but  4 > 3
```

Thus, not all matrices correspond to permutations. Nevertheless, the binary representation does afford a metric in the form of Hamming distance and suggests an alternative way of constructing deceptive ordering problems, since once a binary representation exists several methods for constructing misleading problems could be employed. Deb and Goldberg (1992) explain how to construct trap functions. Whitley (1991) also discusses the construction of deceptive binary functions.

While the topic of deception has been the focus of some controversy (cf Grefenstette 1993), there are few tools for understanding the difficulty or complexity of permutation problems. Whitley and Yoo found that simulation results failed to provide clear evidence that *deceptive* functions built using o-schema fitness averages really were misleading or difficult for simple genetic algorithms.

Aside from the fact that many problems with permutation-based representations are known to be NP complete problems, there is little work which characterizes the complexity of specific instances of these problems, especially from a genetic algorithm perspective. One can attempt to estimate the size and depth of basins of attraction, but such methods must presuppose the use of a particular search methods. The use of different local search operators can induce different numbers of local optima and different sized basins of attraction. Changing representations can have the same effect.

C1.4.8 Operator descriptions and local search

Section C3.2.3 of this handbook, on mutation for permutations, also provides information on local search operators, the most well known of which is 2-opt. Information on the most commonly used forms of recombination for permutation-based representations is found in Section C3.3.3. For a general discussion of permutations, see the books by Niven (1965) and Knuth (1973). Whitley and Yoo (1995) present methods for constructing infinite-population models for simple genetic algorithms applied to permutation problems which can be easily converted into finite-population Markov models.

C3.2.3
C3.3.3

References

Bean J C 1994 Genetic algorithms and random keys for sequencing and optimization *ORSA J. Comput.* **6**

Cormen T, Leiserson C and Rivest R 1990 *Introduction to Algorithms* (Cambridge, MA: MIT Press)

Deb K and Goldberg D 1993 Analyzing deception in trap functions *Foundations of Genetic Algorithms 2* ed D Whitley (San Mateo, CA: Morgan Kaufmann)

Fox B R and McMahon M B 1991 *Genetic Operators for Sequencing Problems Foundations of Genetic Algorithms* ed G J E Rawlins (San Mateo, CA: Morgan Kaufmann) pp 284–300

Franz D R 1972 *Non-Linearities in Genetic Adaptive Search* PhD Dissertation, University of Michigan

Goldberg D 1987 Simple genetic algorithms and the minimal, deceptive problem *Genetic Algorithms and Simulated Annealing* ed L Davis (Boston, MA: Pitman)

——1989 *Genetic Algorithms in Search, Optimization and Machine Learning* (Reading, MA: Addison-Wesley)

Goldberg D and Lingle R Jr 1985 Alleles, loci, and the traveling salesman problem *Proc. 1st Int. Conf. on Genetic Algorithms and Their Applications* ed J J Grefenstette (Hillsdale, NJ: Erlbaum)

Grefenstette J 1993 Deception considered harmful *Foundations of Genetic Algorithms 2* ed D Whitley (San Mateo, CA: Morgan Kaufmann)

Holland J 1975 *Adaptation In Natural and Artificial Systems* (University of Michigan Press)

Kargupta H, Deb K and Goldberg D 1992 Ordering genetic algorithms and deception *Parallel Problems Solving from Nature, 2* ed R Manner and B Manderick (Amsterdam: Elsevier) pp 47–56

Knuth R M 1973 *The Art of Computer Programming Volume 3: Sorting and Searching* (Reading, MA: Addison-Wesley)

Niven I 1965 *Mathematics of Choice, or How to Count without Counting* (Mathematical Association of America)

Oliver I, Smith D and Holland J 1987 A study of permutation crossover operators on the traveling salesman problem *2nd Int. Conf. on Genetic Algorithms (Pittsburgh, PA, 1987)* ed J J Grefenstette (Hillsdale, NJ: Erlbaum) pp 224–30

Radcliffe N 1991 Forma analysis and random respectful recombination *Fourth Int. Conf. on Genetic Algorithms (San Diego, CA, July 1991)* ed R Belew and L Booker (San Mateo, CA: Morgan Kaufmann) pp 222–9

——1993 Genetic set recombination *Foundations of Genetic Algorithms 2* ed D Whitley (San Mateo, CA: Morgan Kaufmann) pp 203–19

Schaffer J D, Caruana R A, Eshelman L J and Das R 1989 A study of control parameters affecting online performance of genetic algorithms for function optimization *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 51–60

Shaefer C 1987 The ARGOT strategy: adaptive representation genetic optimizer technique *2nd Int. Conf. on Genetic Algorithms (Pittsburgh, PA, 1987)* ed J J Grefenstette (Hillsdale, NJ: Erlbaum) pp 50–8

Starkweather T, McDaniel S, Mathias K, Whitley D and Whitley C 1991 A comparison of genetic sequencing operators *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, 1991)* ed R K Belew and L B Booker (San Mateo, CA: Morgan Kauffman)

Syswerda G 1989 Uniform crossover in genetic algorithms *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 2–9

Whitley D 1991 Fundamental principles of deception in genetic search *Foundations of Genetic Algorithms* ed G J E Rawlins (San Mateo, CA: Morgan Kaufmann) pp 221–41

Whitley D and Yoo N-W 1995 Modeling simple genetic algorithms for permutation problems *Foundations of Genetic Algorithms 3* ed D Whitley and M Vose (San Mateo, CA: Morgan Kaufmann) pp 163–84

C1.5 Finite-state representations

David B Fogel

Abstract

Finite-state representations are described and defined. The potential application of these structures in prediction problems is detailed.

C1.5.1 Introduction

A finite-state machine is a mathematical logic. It is essentially a computer program: it represents a sequence of instructions to be executed, each depending on a current state of the machine and the current stimulus. More formally, a finite-state machine is a 5-tuple

$$M = (Q, \tau, \rho, s, o)$$

where Q is a finite set, the set of states, τ is a finite set, the set of input symbols, ρ is a finite set, the set of output symbols, $s : Q \times \tau \rightarrow Q$ is the next state function, and $o : Q \times \tau \rightarrow \rho$ is the next output function.

Any 5-tuple of sets and functions satisfying this definition is to be interpreted as the mathematical description of a machine that, if given an input symbol x while it is in state q, will output the symbol $o(q, x)$ and transition to state $s(q, x)$. Only the information contained in the current state describes the behavior of the machine for a given stimulus. The entire set of states serves as the 'memory' of the machine. Thus a finite-state machine is a transducer that can be stimulated by a finite alphabet of input symbols, that can respond in a finite alphabet of output symbols, and that possesses some finite number of different internal states. The corresponding input–output symbol pairs and next-state transitions for each input symbol, taken over every state, specify the behavior of any finite-state machine, given any starting state. For example, a three-state machine is shown in figure C1.5.1. The alphabet of input symbols are elements of the set $\{0, 1\}$, whereas the alphabet of output symbols are elements of the set $\{\alpha, \beta, \gamma\}$ (input symbols are shown to the left of the slash, output symbols are shown to the right). The finite-state machine transforms a sequence of input symbols into a sequence of output symbols. Table C1.5.1 indicates the response of the machine to a given string of symbols, presuming that the machine is found in state C. It is presumed that the machine acts when each input symbol is perceived and the output takes place before the next input symbol arrives.

Table C1.5.1. The response of the finite-state machine shown in figure C1.5.1 to a string of symbols. In this example, the machine starts in state C.

Present state	C	B	C	A	A	B
Input symbol	0	1	1	1	0	1
Next state	B	C	A	A	B	C
Output symbol	β	α	γ	β	β	α

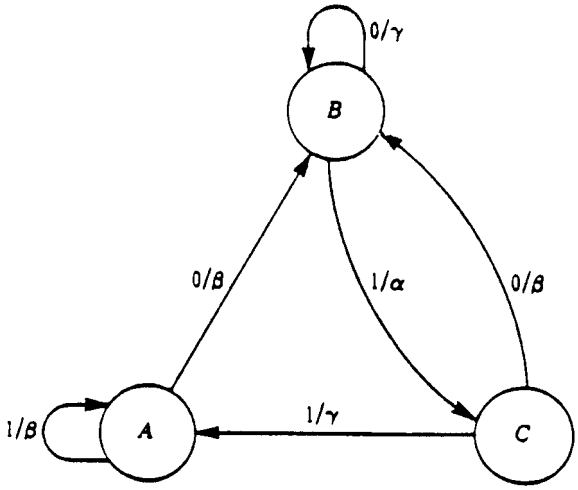


Figure C1.5.1. A three-state finite machine. Input symbols are shown to the left of the slash. Output symbols are to the right of the slash. Unless otherwise specified, the machine is presumed to start in state *A*. (After Fogel *et al* 1966, p 12).

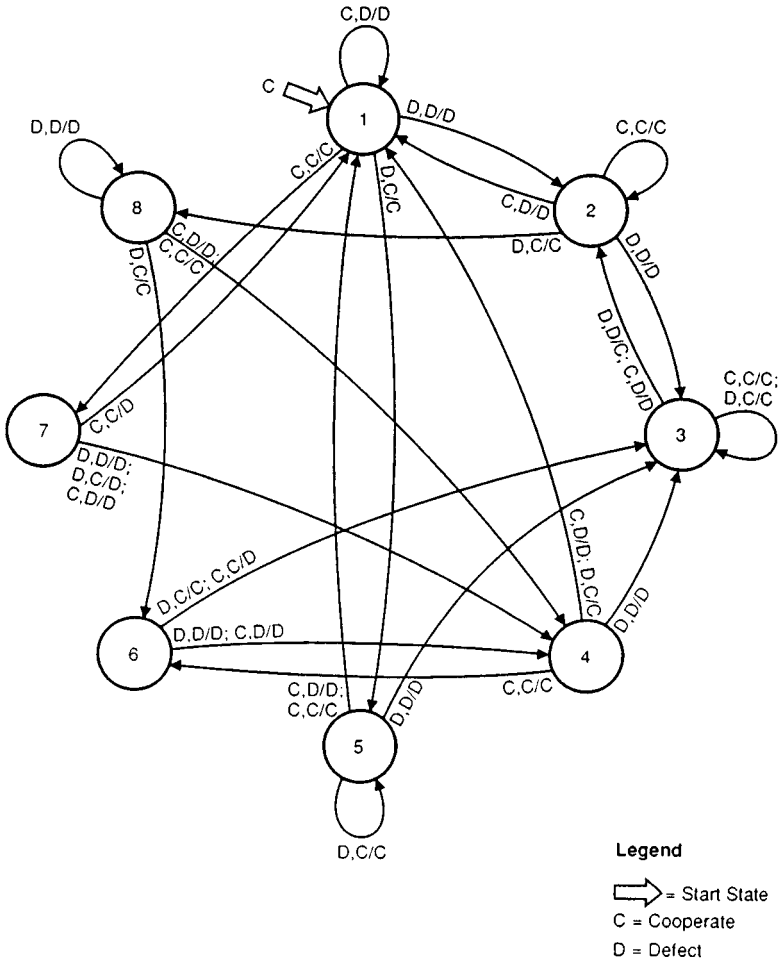


Figure C1.5.2. A finite-state machine evolved in prisoner's dilemma experiments detailed by Fogel (1995b, p 215). The input symbols form the set $\{(C, C), (C, D), (D, C), (D, D)\}$ and the output symbols form the set $\{C, D\}$. The machine also has an associated first move indicated by the arrow; here the machine cooperates initially then proceeds into state 6.

C1.5.2 Applications

Finite-state representations are often convenient when the required solutions to a particular problem of interest require the generation of a sequence of symbols having specific meaning. For example, consider the problem offered by Fogel *et al* (1966) of predicting the next symbol in a sequence of symbols taken from some alphabet A (here, $\tau = \rho = A$). A population of finite-state machines is exposed to the environment, that is, the sequence of symbols that have been observed up to the current time. For each parent machine, as each input symbol is offered to the machine, each output symbol is compared with the next input symbol. The worth of this prediction is then measured with respect to the given payoff function (e.g. all–none, absolute error, squared error, or any other expression of the meaning of the symbols). After the last prediction is made, a function of the payoff for each symbol (e.g. average payoff per symbol) indicates the fitness of the machine. Offspring machines are created through *mutation* and/or *recombination*. The machines that provide the greatest payoff are retained to become parents of the next generation. This process is iterated until an actual prediction of the next symbol (as yet inexperienced) in the environment is required. The best machine generates this prediction, the new symbol is added to the experienced environment, and the process is repeated.

C3.2.4 C3.3.4

There is an inherent versatility in such a procedure. The payoff function can be arbitrarily complex and can possess temporal components; there is no requirement for the classical squared-error criterion or any other smooth function. Further, it is not required that the predictions be made with a one-step look ahead. Forecasting can be accomplished at an arbitrary length of time into the future. Multivariate environments can be handled, and the environmental process need not be stationary because the simulated evolution will adapt to changes in the transition statistics.

For example, Fogel (1991, 1993, 1995a) has used finite-state machines to describe behaviors in the iterated prisoner's dilemma. The input alphabet was selected as the set $\{(C, C), (C, D), (D, C), (D, D)\}$ where C corresponds to a move for cooperation and D corresponds to a move for defection. The ordered pair (X, Y) indicates that the machine played X in the last move, while its opponent played Y. The output alphabet was the set $\{C, D\}$ and corresponded to the next move of the machine based on the previous pair of moves and the current state of the machine (see figure C1.5.2).

Other applications of finite-state representations have been offered. For example, Jefferson *et al* (1991), Angeline and Pollack (1993), and others employed a finite-state machine to describe the behavior of a simulated ant on a trail placed on a grid. The input alphabet was the set $\{0, 1\}$, where 0 indicated that the ant did not see a trail cell ahead and 1 indicated that it did see such a cell ahead. The output alphabet was $\{M, L, R, N\}$ where M indicated a move forward, L indicated a turn to the left without moving, R indicated a turn to the right without moving, and N indicated a condition to do nothing. The task was to evolve a finite-state machine that would generate a sequence of moves to traverse the trail in the shortest number of time steps.

References

Angeline P J and Pollack J B 1993 Evolutionary module acquisition *Proc. 2nd Ann. Conf. on Evolutionary Programming (San Diego, CA)* ed D B Fogel and W Atmar (La Jolla, CA: Evolutionary Programming Society) pp 154–63

Fogel D B 1991 The evolution of intelligent decision-making in gaming *Cybernet. Syst.* **22** 223–36

——1993 Evolving behaviors in the iterated prisoner's dilemma *Evolut. Comput.* **1** 77–97

——1995a On the relationship between the duration of an encounter and the evolution of cooperation in the iterated prisoner's dilemma *Evolut. Comput.* **3** 349–63

——1995b *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence* (Piscataway, NJ: IEEE)

Fogel L J, Owens A J and Walsh M J 1966 *Artificial Intelligence Through Simulated Evolution* (New York: Wiley)

Jefferson D, Collins R, Cooper C, Dyer M, Flowers M, Korf R, Taylor C and Wang A 1991 Evolution as a theme in artificial life: the Genesys/Tracker system *Artificial Life II* ed C G Langton, C Taylor, J D Farmer and S Rasmussen (Reading, MA: Addison-Wesley) pp 549–77

C1.6 Parse trees

Peter J Angeline

Abstract

This section reviews parse tree representations, a popular representation for evolving executable structures. The field of genetic programming is based entirely on the flexibility of this representation. This section describes some of the history of parse trees in evolutionary computation, the form of the representation and some special properties.

When an executable structure such as a program or a function is the object of an evolutionary computation, representation plays a crucial role in determining the ultimate success of the system. If a traditional, syntax-laden programming language is chosen to represent the evolving programs, then manipulation by simple evolutionary operators will most likely produce syntactically invalid offspring. A more beneficial approach is to design the representation to ensure that only syntactically correct programs are created. This reduces the ultimate size of the search space considerably. One method for ensuring syntactic correctness of generated programs is to evolve the desired program's parse tree rather than an actual, unparsed, syntax-laden program. Use of the parse tree representation completely removes the 'syntactic sugar' introduced into a programming language to ensure human readability and remove parsing ambiguity.

Cramer (1985), in the first use of a parse tree representation in a *genetic algorithm*, described two distinct representations for evolving sequential computer programs based on a simple algorithmic language and emphasized the need for offspring programs to remain syntactically correct after manipulation by the genetic operators. To accomplish this, Cramer investigated two encodings of the language into fixed-length integer representations. B1.2

Cramer (1985) first represented a program as an ordered collection of statements. Each statement consisted of N integers; the first integer identified the command to be executed and the remainder specified the arguments to the command. If the command required fewer than $N-1$ arguments, then the trailing integers in the statement were ignored. Depending on the syntax of the statement's command, an integer argument could identify a variable to be manipulated or a statement to be executed. Consequently, even though the program was stored as a sequence it implicitly encoded an execution tree that could be reconstructed by replacing all arguments referring to program statements with the actual statement. Cramer (1985) noted that this representation was not suitable for manipulation by genetic operators and occasionally resulted in infinite loops when two auxiliary statements referred to each other.

The second representation for simple programs reviewed in Cramer (1985) alleviated some of the deficiencies of the first by making the implicit tree representation explicit. Instead of evolving a sequence of statements with arguments that referred to other statements, this representation replaces these arguments with the actual statement. For instance, an encoded program would have the form (0 (3 5) (1 3 (1 4 (4 5)))) where a matching set of parentheses denotes a single complete statement. Note that in the language used by Cramer (1985), a subtree argument does not return a value to the calling statement but only designates a command to be executed.

Probably the best known use of the parse tree representation is that by Koza (1992), an example of which is shown in figure C1.6.1. The only difference between the representations used in *genetic programming* (Koza 1992) and the explicit parse tree representation (Cramer 1985) is that the subtree arguments in genetic programming return values to their calling statements. This provides a direct mechanism for the communication of intermediate values to other portions of the parse tree representation and fortifies a subtree as an independent computational unit. The variety of problems investigated by Koza (1992) demonstrates the flexibility and applicability of this representational paradigm. B1.5.1

An appealing aspect of the parse tree representation is its natural recursive definition, which allows for dynamically sized structures. All parse tree representations investigated to date have included an associated restriction on the size of the evolving programs. Without such a restriction, the natural dynamics of evolutionary systems would continually increase the size of the evolving programs, eventually swamping the available computational resources. Size restrictions take on two distinct forms. Depth limitation restricts the size of evolving parse trees based on a user-defined maximal depth parameter. Node limitation places a limit on the total number of nodes available for an individual parse tree. Node limitation is the preferred method of the two since it encodes fewer restrictions on the structural organization of the evolving programs (Angeline 1996).

In a parse tree representation, the primitive language—the contents of the parse tree—determines the power and suitability of the representation. Sometimes the elements of this language are taken from existing programming languages, but typically it is more prudent to design the primitive language so that it takes into consideration as much domain-specific knowledge as available. Failing to select language primitives tailored to the task at hand may prevent the acquisition of solutions. For instance, if the objective is to evolve a function that has a particular periodic behavior, it is important to include base language primitives that also have periodic behavior, such as the mathematical functions $\sin x$ and $\cos x$.

Due to the acyclic structure of parse trees, iterative computations are often not naturally represented. It is often difficult for an evolutionary computation to correctly identify appropriate stopping criteria for loop constructs introduced into the primitive language. To compensate, the evolved function often is evaluated within an implied 'repeat until done' loop that reexecutes the evolved function until some predetermined stopping criterion is satisfied. For instance, Koza (1992) describes evolving a controller for an artificial ant for which the fitness function repeatedly applies its program until a total of 400 commands are executed or the ant completes the task. Numerous examples of such implied loops can be found in the genetic programming literature (e.g. Koza 1992, pp 147, 329, 346, Teller 1994, Reynolds 1994, Kinnear 1993).

Often it is necessary to include constants in the primitive language, especially when mathematical expressions are being evolved. The general practice is to include as a potential terminal of the language a special symbol that denotes a constant. When a new individual is created and this symbol is selected to be a terminal, rather than enter the symbol into the parse tree, a numerical constant is inserted drawn uniformly from a user-defined range (Koza 1992). Figure C1.6.1 shows a number of numerical constants that would be inserted into the parse tree in this manner.

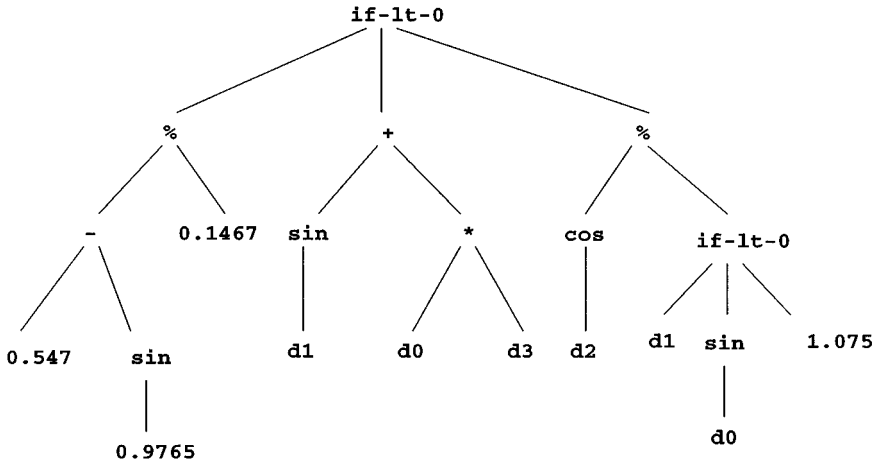


Figure C1.6.1. An example parse tree representation for a complex numerical function. The function if-lt-0 is a numerical conditional that returns the value of its second argument if its first argument evaluates to a negative number and otherwise returns the value of its third argument. The function % denotes a protected division operator that returns a value of 1.0 if the second argument (the denominator) is zero.

Typically, the language defined for a parse tree representation is syntactically homogenous, meaning that the return values of all functions and terminals are the same computational type, (e.g. integer). Montana (1995) has investigated the evolution of multityped parse trees and shown that extra syntactic

considerations do not drastically increase the complexity of the associated genetic operators. Koza (1992) also investigates constrained parse tree representations.

Given the recursive nature of parse trees, they are a natural representation in which to investigate issues about inducing modular structures. Currently, three methods for inducing modular parse trees have been proposed. Angeline and Pollack (1994) add two mutation operators to their Genetic Program Builder (GLiB) system, which dynamically form and destroy modular components out of the parse tree. The *compress* mutation operation, which bears some resemblance to the *encapsulate* operator of Koza (1992), selects a subtree and makes it a new representational primitive in the language. The *expand* mutation operation reverses the actions of the compress mutation by selecting a compressed subtree in the individual and replacing it with the original subtree. Angeline and Pollack (1994) claim that the natural evolutionary dynamics of the genetic program automatically discover effective modularizations of the evolving programs. Rosca and Ballard (1996) with their Adaptive Representation method use a set of heuristics to evaluate the usefulness of all subtrees in the population and then create subroutines from the ones that are most useful. Koza (1994) describes a third method for creating modular programs, called *automatically defined functions* (ADFs), which allow the user to determine the number of subroutines to which the main program can refer. During evolution, the definitions of both the main routine and all of its subroutines are evolved in parallel. Koza and Andre (1996) have more recently included a number of mutations to dynamically modify various aspects of ADFs in order to reduce the amount of prespecification required by the user.

B1.5.1, H1.1.3

References

Angeline P J 1996 Genetic programming's continued evolution *Advances in Genetic Programming* vol 2, ed P J Angeline and K Kinnear (Cambridge, MA: MIT Press) pp 1–20

Angeline P J and Pollack J B 1994 Co-evolving high-level representations *Artificial Life III* ed C G Langton (Reading, MA: Addison-Wesley) pp 55–71

Cramer N L 1985 A representation for the adaptive generation of simple sequential programs *Proc. 1st Int. Conf. on Genetic Algorithms (Pittsburgh, PA, July 1985)* ed J J Grefenstette (Hillsdale, NJ: Erlbaum) pp 183–7

Kinnear K E 1993 Generality and difficulty in genetic programming: evolving a sort *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 287–94

Koza J R 1992 *Genetic Programming: on the Programming of Computers by Means of Natural Selection* (Cambridge, MA: MIT Press)

——1994 *Genetic Programming II: Automatic Discovery of Reusable Programs* (Cambridge, MA: MIT Press)

Koza J R and Andre D 1996 Classifying protein segments as transmembrane domains using architecture-altering operations in genetic programming *Advances in Genetic Programming* vol 2, ed P J Angeline and K Kinnear (Cambridge, MA: MIT Press) pp 155–76

Montana D J 1995 Strongly typed genetic programming *Evolutionary Comput.* **3** 199–230

Reynolds C W 1994 Evolution of obstacle avoidance behavior: using noise to promote robust solutions *Advances in Genetic Algorithms* ed K Kinnear (Cambridge, MA: MIT Press) pp 221–43

Rosca J P and Ballard D H 1996 Discovery of subroutines in genetic programming *Advances in Genetic Programming* vol 2, ed P J Angeline and K Kinnear (Cambridge, MA: MIT Press) pp 177–202

Teller A 1994 The evolution of mental models *Advances in Genetic Algorithms* ed K Kinnear (Cambridge, MA: MIT Press) pp 199–220

C1.7 Guidelines for a suitable encoding

David B Fogel and Peter J Angeline

Abstract

In this section, we offer guidelines for choosing a representation. Consideration is given to the interaction between the representation, variation operators, selection method, and objective function. Suitable encodings are seen to be problem dependent.

In any evolutionary computation application to an optimization problem, the human operator determines at least four aspects of the approach: representation, variation operators, method of selection, and objective function. It could be argued that the most crucial of these four is the objective function because it defines the purpose of the operator in quantitative terms. Improperly specifying the objective function can lead to generating the right answer to the wrong problem. However it should be clear that the selections made for each of these four aspects depend in part on the choices made for all the others. For example, the objective function cannot be specified in the absence of a problem representation. The choice for appropriate representation, however, cannot be made in the absence of anticipating the variation operators, the selection function, and the mathematical formulation of the problem to be solved. Thus, an iterative procedure for adjusting the representation and search and selection procedures in light of a specified objective function becomes necessary in many applications of evolutionary computation. This section focuses on selecting the representation for a problem, but it is important to remain cognizant of the interdependent nature of these operations within any evolutionary computation.

There have been proposals that the most suitable encoding for any problem is a *binary encoding* C1.2 because it maximizes the number of schemata being searched implicitly (Holland 1975, Goldberg 1989), but there have been many examples in the evolutionary computation literature where alternative representations have provided for algorithms with greater efficiency and optimization effectiveness when compared with identical problems (see e.g. the articles by Bäck and Schwefel (1993) and Fogel and Stayton (1994) among others). Davis (1991) and Michalewicz (1996) comment that in many applications *real-valued* or C1.3 other representations may be chosen to advantage over binary encodings. There does not appear to be any general benefit to maximizing implicit parallelism in evolutionary algorithms, and, therefore, forcing problems to fit binary representation is not recommended.

The close relationship between representation and other facets of evolutionary computation suggests that, in many cases, the appropriate choice of representation arises from the operator's ability to visualize the dynamics of the resulting search on an adaptive landscape. For example, consider the problem of finding the minimum of the quadratic surface

$$f(x, y) = x^2 + y^2 \qquad x, y \in \mathbb{R}.$$

Immediately, it is easy to visualize this function as shown in figure C1.7.1. If an evolutionary approach to the problem were to be taken, an intuitive representation suggested by the surface is to operate on the real values of (x, y) directly (rather than recoding these values into some other alphabet). Accordingly, a reasonable choice of variation operator would be the imposition of a continuous random perturbation to each dimension (x, y) (perhaps a zero-mean Gaussian perturbation as is common in evolution strategies and evolutionary programming). This would be followed by a hard selection against all but the best solution in the current population, given that the function is strongly convex. With even slight experience, the resulting population dynamics of this approach can be visualized without executing a single line of code. In contrast, for this problem other representational choices and variation operators (e.g. mapping

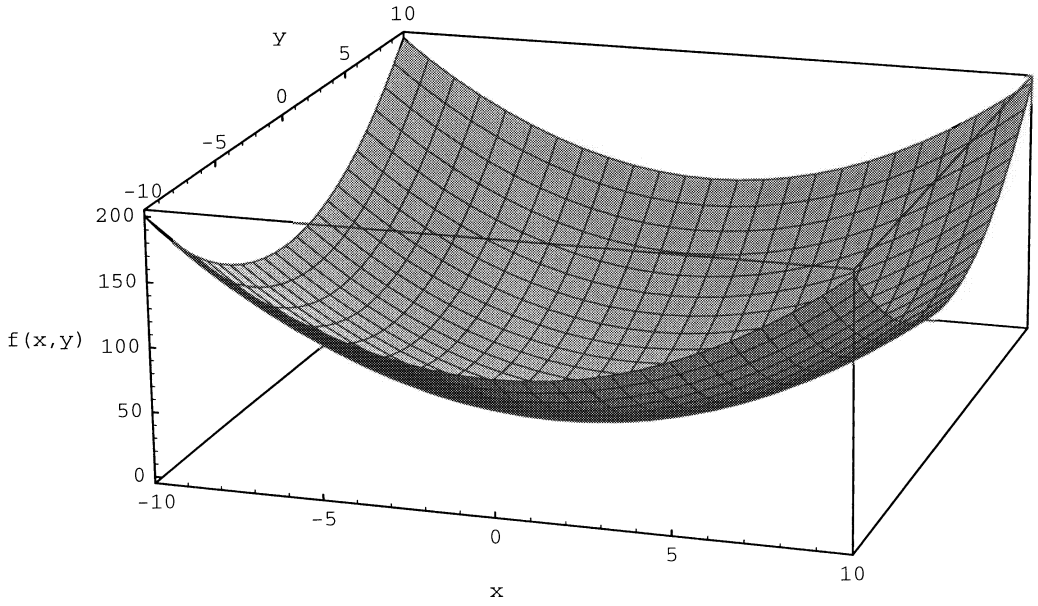


Figure C1.7.1. A quadratic bowl in two dimensions. The shape of the response surface suggests a natural approach for optimization. The intuitive choice is to use real-valued encodings and continuous variation operators. The shape of a response surface can be useful in suggesting choices for suitable encodings.

the real numbers into binary and then applying crossover operators to the binary encoding) are contrived, difficult to visualize, and appear more likely to be ineffectual (see Schraudolph and Belew 1992, Fogel and Stayton 1994).

Thus, the basic recommendation for choosing a suitable encoding is that the representation should be suggested from the problem at hand. If a *traveling salesman problem* is to be investigated, obvious natural choices for the encoding are a list of cities to be visited in order, or a corresponding list of edges. For a discrete-symbol time-series prediction problem, *finite-state machines* may be especially appropriate. For continuous time-series prediction, other model forms (e.g. neural networks, ARMA, or Box–Jenkins) appear better suited. In nonstationary environments, that is, fitness functions that are dynamic rather than static, it is often necessary to include some form of memory in the representation. Diplodic representations—representations that include two alleles per gene—have been used to model cyclic environments (Goldberg and Smith 1987, Ng and Wong 1995). The most natural choice for representation is a subjective choice, and it will differ across investigators, although, like a suitable scientific model, a suitable representation should be as complex as necessary (and no more so) and should 'explain' the phenomena investigated, which here means that the resulting search should be visualizable or imaginable to some extent.

G9.5 C1.5

References

Bäck T and Schwefel H-P 1993 An overview of evolutionary algorithms for parameter optimization *Evolutionary Comput.* **1** 1–24

Davis L (ed) 1991 *Handbook of Genetic Algorithms* (New York: Van Nostrand Reinhold)

Fogel D B and Stayton L C 1994 On the effectiveness of crossover in simulated evolutionary optimization *BioSystems* **32** 171–82

Goldberg D E 1989 *Genetic Algorithms in Search, Optimization, and Machine Learning* (Reading, MA: Addison-Wesley)

Goldberg D E and Smith R E 1987 Nonstationary function optimization using genetic algortihms with dominance and diploidy *Proc. 2nd Int. Conf. on Genetic Algorithms (Cambridge, MA, 1987)* ed J J Grefenstette (Hillsdale, NJ: Erlbaum) pp 59–68

Holland J H 1975 *Adaptation in Natural and Artificial Systems* (Ann Arbor, MI: University of Michigan Press)

Michalewicz Z 1996 *Genetic Algorithms + Data Structures = Evolution Programs* 3rd edn (Berlin: Springer)

Ng K P and Wong K C 1995 A new diploid scheme and dominance change mechanism for non-stationary function optimization *Proc. 6th Int. Conf. on Genetic Algorithms (Pittsburgh, PA, July 1995)* ed L J Eshelman (San Mateo, CA: Morgan Kaufmann) pp 159–66

Schraudolph N N and Belew R K 1992 Dynamic parameter encoding for genetic algorithms *Machine Learning* **9** 9–21

C1.8 Other representations

Peter J Angeline and David B Fogel

Abstract

In this section, nonstandard representations are considered and detailed. Attention is devoted to mixed-integer structures, as well as structures that incorporate introns (i.e. noncoding segments) and diploid (double-stranded) representations.

C1.8.1 Mixed-integer structures

Many real-world applications suggest the use of representations that are hybrids of the canonical representations. One common instance is the simultaneous use of discrete and continuous object variables, with a general formulation of the global optimization problem as follows (Bäck and Schütz 1995):

$$\min\{f(\boldsymbol{x}, \boldsymbol{d}) | \boldsymbol{x} \in M, R^n \supseteq M, \boldsymbol{d} \in N, Z^{n_d} \supseteq N\}.$$

Within *evolution strategies* and *evolutionary programming*, the common representation is simply the real-integer vector pair (i.e. no effort is made to encode these vectors into another representation such as binary). Sections C3.2.6 and C3.3.6 offer methods for mutating and recombining the above representations. B1.3, B1.4 C3.2.6, C3.3.6

Mixed representations also occur in the application of evolutionary algorithms to *neural networks* or *fuzzy logic systems*, where real-world parameters are used to define weights or shapes of membership functions and integer values are used to define the number of nodes and their connections, or the number of membership functions (see e.g. Fogel 1995, Angeline *et al* 1994, McDonnell and Waagen 1994, Haffner and Sebald 1993). D1 D2

C1.8.2 Introns

In contrast to the above hybridization of different forms of representation, another 'nontraditional' approach has involved the inclusion of noncoding regions (introns) within a solution (see e.g. Levenick 1991, Golden *et al* 1995, Wu and Lindsay 1995). Solutions are represented in the form

$$x_1|\text{intron}|x_2|\text{intron}|\ldots|\text{intron}|x_n$$

where there are n components to vector $\boldsymbol{x}$. Introns have been hypothesized to allow for greater efficiency in recombining building blocks (see Section C3.3.6). C3.3.6

In the standard genetic algorithm representation, the semantics of an allele value (how the allele is interpreted) is usually tied to its position in the fixed-length n-ary string. For instance, in a binary string representation, each position signifies the presence or absence of a specific feature in the genome being decoded. The difficulty with such a representation is that with positions in the string representation that are semantically linked but separated by a large number of intervening positions in the string crossover has a high probability of disrupting beneficial settings for these two positions. Goldberg *et al* (1989) describe a representation for a genetic algorithm that embodies one approach to addressing this problem. In their *messy genetic algorithm* (mGA), each allele value is represented as a pair of values, one specifying the actual allele value and one specifying the position the allele occupies. Messy GAs are defined to be of variable length, and Goldberg *et al* (1989) describe appropriate methods for resolving underdetermined or overdetermined genomes. In this representation it is important to note that the semantics are literally carried along with the allele value in the form of the allele's string position. C4.2.4

C1.8.3 Diploid representations

Diploid representations, representations that include multiple allele values for each position in the genome, have been offered as mechanisms for modeling cyclic environments. In a diploid representation, a method for determining which allele value for a gene will be expressed is required to adjudicate when the allele values do not agree. Building on earlier investigations (see e.g. Bagley 1967, Hollstein 1971, Brindle 1981) Goldberg and Smith (1987) demonstrate that an evolving dominance map allows quicker adaptation to cyclical environment changes than either a haploid representation or a diploid representation using a fixed dominance mapping. Goldberg and Smith (1987) use a triallelic representation from Hollstein (1971): 1, i, and 0. Both 1 and i map to the allele value of '1', while 0 maps to the allele value of '0' with 1 dominating both i and 0 and 0 dominating i. Thus, the dominance of a 1 over a 0 allele value could be altered via mutation by altering the value to an i. Ng and Wong (1995) extend the multiallele approach to dominance computation by adding a fourth value for a recessive 0. Thus 1 dominates 0 and o while 0 dominates i and o. When both allele values for a gene are dominant or recessive, then one of the two values is chosen randomly to be the dominant value. Ng and Wong (1995) also suggest that the dominance of all of the components in the genome should be reversed when the fitness value of an individual falls by 20% or more between generations.

References

Angeline P J, Saunders G M and Pollack J B 1994 An evolutionary algorithm that constructs recurrent neural networks *IEEE Trans. Neural Networks* **NN-5** 54–65

Bäck T and Schütz M 1995 Evolution strategies for mixed-integer optimization of optical multilayer systems *Proc. 4th Ann. Conf. on Evolutionary Programming (San Diego, CA, March 1995)* ed J R McDonnell, R G Reynolds and D B Fogel (Cambridge, MA: MIT Press) pp 33–51

Bagley J D 1967 *The Behavior of Adaptive Systems which Employ Genetic and Correlation Algorithms* Doctoral Dissertation, University of Michigan; University Microfilms 68-7556

Brindle A 1981 *Genetic Algorithms for Function Optimization* Doctoral Dissertation, University of Alberta

Cobb H G and Grefenstette J J 1993 Genetic algorithms for tracking changing environments *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 523–30

Fogel D B 1995 *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence* (Piscataway, NJ: IEEE)

Goldberg D E, Korb D E and Deb K 1989 Messy genetic algorithms: motivation, analysis, and first results *Complex Syst.* **3** 493–530

Goldberg D E and Smith R E 1987 Nonstationary function optimization using genetic algorithms with dominance and diploidy *Proc. 2nd Int. Conf. on Genetic Algorithms (Cambridge, MA, July 1987)* ed J J Grefenstette (Hillsdale, NJ: Erlbaum) pp 59–68

Golden J B, Garcia E and Tibbetts C 1995 Evolutionary optimization of a neural network-based signal processor for photometric data from an automated DNA sequencer *Proc. 4th Ann. Conf. on Evolutionary Programming (San Diego, CA, March 1995)* ed J R McDonnell, R G Reynolds and D B Fogel (Cambridge, MA: MIT Press) pp 579–601

Haffner S B and Sebald A V 1993 Computer-aided design of fuzzy HVAC controllers using evolutionary programming *Proc. 2nd Ann. Conf. on Evolutionary Programming (San Diego, CA, 1993)* ed D B Fogel and W Atmar (La Jolla, CA: Evolutionary Programming Society) pp 98–107

Hollstein R B 1971 *Artificial Genetic Adaptation in Computer Control Systems* Doctoral Dissertation, University of Michigan; University Microfilms 71-23, 773

Levenick J R 1991 Inserting introns improves genetic algorithm success rate: taking a cue from biology *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, July 1991)* ed R K Belew and L B Booker (San Mateo, CA: Morgan Kaufmann) pp 123–27

McDonnell J R and Waagen D 1994 Evolving recurrent perceptrons for time-series modeling *IEEE Trans. Neural Networks* **NN-5** 24–38

Ng K P and Wong K C 1995 A new diploid scheme and dominance change mechanism for non-stationary function optimization *Proc. 6th Int. Conf. on Genetic Algorithms (Pittsburgh, PA, July 1995)* ed L J Eshelman (San Mateo, CA: Morgan Kaufmann) pp 159–66

Wu A S and Lindsay R K 1995 Empirical studies of the genetic algorithm with noncoding segments *Evolutionary Comput.* **3** 121–48

C2.1 Introduction

Kalyanmoy Deb

Abstract

In this section, an introduction to different selection operators used in evolutionary computation (EC) studies is presented. Different EC methods use different selection operators. However, the working principle of selection operators is discussed by presenting a generic pseudocode. A parameter—selective pressure—that characterizes the selection operators is discussed next. A number of different selection operators are discussed in detail in the subsequent sections.

C2.1.1 Working mechanisms

Selection is one of the main operators used in evolutionary algorithms. The primary objective of the selection operator is to *emphasize* better solutions in a population. This operator does not create any new solution, instead it selects relatively good solutions from a population and deletes the remaining, not-so-good, solutions. Thus, the selection operator is a mix of two different concepts—reproduction and selection. When one or more copies of a good solution are reproduced, this operation is called reproduction. Multiple copies of a solution are placed in a population by deleting some inferior solutions. This concept is known as selection. Although some EC studies use both these concepts simultaneously, some studies use them separately.

The identification of good or bad solutions in a population is usually accomplished according to a solution's fitness. The essential idea is that a solution having a better fitness must have a higher probability of selection. However, selection operators differ in the way the copies are assigned to better solutions. Some operators sort the population according to fitness and deterministically choose the best few solutions, whereas some operators assign a probability of selection to each solution according to fitness and make a copy using that probability distribution. In the probabilistic selection operator, there is some finite, albeit small, probability of rejecting a good solution and choosing a bad solution. However, a selection operator is usually designed in a way so that the above is a low-probability event. There is, of course, an advantage of allowing this stochasticity (or flexibility) in the evolutionary algorithms. Due to a small initial population or an improper parameter choice or in solving a complex nonlinear fitness function, the best few individuals in a finite population may sometimes represent a suboptimal region. If a deterministic selection operator is used, these seemingly good individuals in the population will be emphasized and the population may finally converge to a wrong solution. However, if a stochastic selection operator is used, diversity in the population will be maintained by occassionally choosing not-so-good solutions. This event may prevent EC algorithms from making a hasty decision in converging to a wrong solution.

In the following, we present a pseudocode for the selection operator and then discuss briefly some of the popular selection operators.

C2.1.2 Pseudocode

Some EC algorithms (specifically, *genetic algorithms* (GAs) and *genetic programming* (GP)) usually apply the selection operator first to select good solutions and then apply the recombination and mutation operators on these good solutions to create a hopefully better set of solutions. Other EC algorithms (specifically, B1.2, B1.5.1

evolution strategies (ES) and *evolutionary programming* (EP)) prefer using the recombination and mutation operator first to create a set of solutions and then use the selection operator to choose a good set of solutions. The selection operator in $(\mu + \lambda)$ ES and EP techniques chooses the offspring solutions from a combined population of parent solutions and solutions obtained after recombination and mutation. In the case of EP, this is done statistically. However, the selection operator in (μ, λ) ES chooses the offspring solutions only from the solutions obtained after the recombination and mutation operators. Since the selection operators are different in different EC studies, it is difficult to present a common code for all selection operators. However, the following pseudocode is a generic for most of the selection operators used in EC studies. B1.3, B1.4

The parameters μ and λ are the numbers of parent solutions and offspring solutions after recombination and mutation operators, respectively. The parameter q is a parameter related to the operator's selective pressure, a matter we discuss later in this section. The population at iteration t is denoted by $P(t) = \{\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots\}$ and the population obtained after the recombination and mutation operators is denoted by $P'(t) = \{\boldsymbol{a}'_1, \boldsymbol{a}'_2, \ldots\}$. Since GAs and GP techniques use the selection operator first, the population $P'(t)$ before the selection operation is an empty set, with no solutions. The fitness function is represented by $\boldsymbol{F}(t)$.

Input: μ, λ, q, $P(t) \in I^{\mu}$, $P'(t) \in I^{\lambda}$, $\boldsymbol{F}(t)$

Output: $P''(t) = \{\boldsymbol{a}''_1, \boldsymbol{a}''_2 \ldots, \boldsymbol{a}''_{\mu}\} \in I^{\mu}$

1 **for** $i \leftarrow 1$ **to** μ
  $\boldsymbol{a}''_i(t) \leftarrow s_{\text{selection}}(P(t), P'(t), \boldsymbol{F}(t), q)$;
2 **return**$(\{\boldsymbol{a}''_1(t), \ldots, \boldsymbol{a}''_{\mu}(t)\})$;

Detailed discussions of some of the selection operators are presented in the subsequent sections. Here, we outline a brief introduction to some of the popular selection schemes, mentioned as $s_{\text{selection}}$ in the above pseudocode.

In the *proportionate* selection operator, the expected number of copies a solution receives is assigned proportionally to its fitness. Thus, a solution having twice the fitness of another solution receives twice as many copies. The simplest form of the proportionate selection scheme is known as the roulette-wheel selection, where each solution in the population occupies an area on the roulette wheel proportional to its fitness. Then, conceptually, the roulette wheel is spun as many times as the population size, each time selecting a solution marked by the roulette-wheel pointer. Since the solutions are marked proportionally to their fitness, a solution with a higher fitness is likely to receive more copies than a solution with a low fitness. There exists a number of variations to this simple selection scheme, which are discussed in Section C2.2. However, one limitation of the proportionate selection scheme is that since copies are assigned proportionally to the fitness values, negative fitness values are not allowed. Also, this scheme cannot handle minimization problems directly. (Minimization problems must be transformed to an equivalent maximization problem in order to use this operator.) Selecting solutions proportional to their fitness has two inherent problems. If a population contains a solution having exceptionally better fitness than the rest of the solutions in the population, this so-called *supersolution* will occupy most of the roulette-wheel area. Thus, most spinning of the roulette wheel is likely to choose the same supersolution. This may cause the population to lose genetic diversity and cause the algorithm to prematurely converge to a suboptimal solution. The second inherent difficulty may arise later in a simulation run, when most of the population members have more or less the same fitness. In this case, the roulette wheel is marked almost equally for each solution in the population and every solution becomes equally likely to be selected. This has the effect of a random selection. Both these inherent difficulties can be avoided by using a *scaling* scheme, where every solution fitness is linearly mapped between a lower and an upper bound before marking the roulette wheel (Goldberg 1989). This allows the selection operator to assign a controlled number of copies, thereby eliminating both the above problems of too large and random assignments. We discuss this scaling scheme further in the next section. Although this selection scheme has been mostly used with GAs and GP applications, in principle it can also be used with both multimembered ES and EP techniques. C2.2

In the *tournament* selection operator, both the scaling problems mentioned above are eliminated by playing tournaments among a specified number of parent solutions according to fitness of solutions. In a tournament of q solutions, the best solution is selected either deterministically or probabilistically. After the tournament is played, there are two options—either all participating q solutions are replaced into the population for the next tournament or they are not replaced until a certain number of tournaments have

been played. In its simplest form (called the binary tournament selection), two solutions are picked and the better solution is chosen. One advantage of this selection method is that this scheme can handle both minimization and maximization problems without any structural change in the fitness function. Only the solution having either the highest or the lowest objective function value need to be chosen depending on whether the problem is a maximization or a minimization problem. Moreover, it has no restriction on negative objective function values. An added advantage of this scheme is that it is ideal for a parallel implementation. Since only a few solutions are required to be compared at a time without resorting to calculation of the population average fitness or any other population statistic, all solutions participating in a tournament can be sent to one processor. Thus, tournaments can be played in parallel on multiple processors and the complete selection process may be performed quickly. Because of these properties, tournament selection is fast becoming a popular selection scheme in most EC studies. Tournament selection is discussed in detail in Section C2.3. C2.3

The *ranking* selection operator is similar to proportionate selection except that the solutions are ranked according to descending or ascending order of their fitness values depending on whether it is a maximization or minimization problem. Each solution is assigned a ranked fitness based on its rank in the population. Thereafter, copies are allocated with the resulting selection probabilities of the solutions calculated using the ranked fitness values. Like tournament selection, this selection scheme can also handle negative fitness values. There exists a number of other schemes based on the concept of the ranking of solutions; these are discussed in Section C2.4. C2.4

In the *Boltzmann* selection operator, a modified fitness is assigned to each solution based on a Boltzmann probability distribution: $\mathcal{F}_i = 1/(1 + \exp(F_i/T))$, where T is a parameter analogous to the temperature term in the Boltzmann distribution. This parameter is reduced in a predefined manner in successive iterations. Under this selection scheme, a solution is selected based on the above probability distribution. Since a large value of T is used initially, almost any solution is equally likely to be selected, but, as the iterations progress, the parameter T becomes small and only good solutions are selected. We discuss this selection scheme further in Section C2.5. C2.5

In the $(\mu + \lambda)$ ES, the selection operator selects μ best solutions deterministically from a pool of all μ parent solutions and λ offspring solutions. Since all parent and offspring solutions are compared, if performed deterministically, this selection scheme guarantees preservation of the best solution found in any iteration.

On the other hand, in the (μ, λ) ES, the selection operator chooses μ best solutions from λ (usually $\lambda > \mu$) offspring solutions obtained by the recombination and mutation operators. Unlike the $(\mu + \lambda)$ ES selection scheme, the best solution found in any iteration is not guaranteed to be preserved throughout a simulation. However, since many offspring solutions are created in this scheme, the search is more exhaustive than that in the $(\mu + \lambda)$ ES scheme. In most applications of the (μ, λ) ES selection scheme, a deterministic selection of best μ solutions is adopted.

In modern variants of the EP technique, a slightly different selection scheme is used. In a pool of parent (of size μ) and offspring solutions (of size the same as the parent population size), each solution is first assigned a score depending on how many solutions it is better than from a set of random solutions (of size q) chosen from the pool. The complete pool is then sorted in descending order of this score and the first μ solutions are chosen deterministically. Thus, this selection scheme is similar to the $(\mu + \mu)$ ES selection scheme with a tournament selection of q tournament size. Bäck *et al* (1994) analyzed this selection scheme as a combination of $(\mu + \mu)$ ES and tournament selection schemes, and found some convergence characteristics of this operator.

Goldberg and Deb (1991) have compared a number of popular selection schemes in terms of their convergence properties, selective pressure, takeover times, and growth factors, all of which are important in the understanding of the power of different selection schemes used in GA and GP studies. Similar studies have also been performed by Bäck *et al* (1994) for selection schemes used in ES and EP studies. A detailed discussion of some analytical as well as experimental comparisons of selection schemes is presented in Section C2.8. In the following section, we briefly discuss the theory of selective pressure and its importance in choosing a suitable selection operator for a particular application. C2.8

C2.1.3 Theory of selective pressure

Selection operators are characterized by a parameter known as the *selective pressure*, which relates to the takeover time of the selection operator. The takeover time is defined as the speed at which the best solution

in the initial population would occupy the complete population by repeated application of the selection operator alone (Bäck 1994, Goldberg and Deb 1991). If the takeover time of a selection operator is *large* (that is, the operator takes a large number of iterations for the best solution to take over the population), the selective pressure of the operator is *small*, and vice versa. Thus, the selective pressure or the takeover time is an important parameter for successful operation of an EC algorithm (Bäck 1994, Goldberg *et al* 1993). This parameter gives an idea of how greedy the selection operator is in terms of making the population uniform with one particular solution. If a selection operator has a large selective pressure, the population loses diversity in the population quickly. Thus, in order to avoid premature convergence to a wrong solution, either a large population is required or highly disruptive recombination and mutation operators are needed. However, a selection operator with a small selection pressure makes a slow convergence and permits the recombination and mutation operators enough iterations to properly search the space. Goldberg and Deb (1991) have calculated takeover times of a number of selection operators used in GAs and GP studies and Bäck (1994) has calculated the takeover time for a number of selection operators used in ES, EP, and GA studies. The former study has also introduced two other parameters—early and late growth rate—characterizing the selection operators.

The growth rate is defined as the ratio of the number of the best solutions in two consecutive iterations. Since most selection operators have different growth rates as the iterations progress, two different growth rates—early and late growth rates—are defined. The early growth rate is calculated initially, when the proportion of the best solution in the population is negligible. The late growth rate is calculated later, when the proportion of the best solution in the population is large (about 0.5). The early growth rate is important, especially if a quick near-optimizer algorithm is desired, whereas the late growth rate can be a useful measure if precision in the final solution is important. Goldberg and Deb (1991) have calculated these growth rates for a number of selection operators used in GAs. A comparison of different selection schemes based on some of the above criteria is given in Section C2.8. C2.8

The above discussion suggests that, for a successful EC simulation, the required selection pressure of a selection operator depends on the recombination and mutation operators used. A selection scheme with a large selection pressure can be used, but only with highly disruptive recombination and mutation operators. Goldberg *et al* (1993) and later Thierens and Goldberg (1993) have found functional relationships between the selective pressure and the probability of crossover for successful working of selectorecombinative GAs. These studies show that a large selection pressure can be used but only with a large probability of crossover. However, if a reasonable selection pressure is used, GAs work successfully for a wide variety of crossover probablities. Similar studies can also be performed with ES and EP algorithms.

References

Bäck T 1994 Selective pressure in evolutionary algorithms: a characterization of selection mechanisms *Proc. 1st IEEE Conf. on Evolutionary Computation (Orlando, FL, 1994)* (Piscataway, NJ: IEEE) pp 57–62

Bäck T, Rudolph G and Schwefel H-P 1994 Evolutionary programming and evolution strategies: similarities and differences *Proc. 2nd Ann. Conf. on Evolutionary Programming (San Diego, CA, July 1994)* ed D B Fogel and W Atmar (La Jolla, CA: Evolutionary Programming Society)

Goldberg D E 1989 *Genetic Algorithms in Search, Optimization, and Machine Learning* (Reading, MA: Addison-Wesley)

Goldberg D E and Deb K 1991 A comparison of selection schemes used in genetic algorithms *Foundations of Genetic Algorithms (Bloomington, IN)* ed G J E Rawlins (San Mateo, CA: Morgan Kaufmann) pp 69–93

Goldberg D E, Deb K and Theirens D 1993 Toward a better understanding of mixing in genetic algorithms *J. SICE* **32** 10–6

Thierens D and Goldberg D E 1993 Mixing in genetic algorithms *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 38–45

C2.2 Proportional selection and sampling algorithms

John Grefenstette

Abstract

Proportional selection assigns to each individual a reproductive probability that is proportional to the individual's relative fitness. This section presents the proportional selection method as a series of steps: (i) map the objective function to fitness, (ii) create a probability distribution proportional to fitness, and (iii) draw samples from this distribution. Characterizations of selective pressure, fitness scaling techniques, and alternative sampling algorithms are also presented.

C2.2.1 Introduction

Selection is the process of choosing individuals for reproduction in an evolutionary algorithm. One popular form of selection is called *proportional selection*. As the name implies, this approach involves creating a number of offspring in proportion to an individual's fitness. This approach was proposed and analyzed by Holland (1975) and has been used widely in many implementations of evolutionary algorithms. C2.1

Besides having some interesting mathematical properties, proportional selection provides a natural counterpart in artificial evolutionary systems to the usual practice in population genetics of defining an individual's fitness in terms of its number of offspring.

For clarity of discussion, it is convenient to decompose the selection process into distinct steps, namely:

(i) map the objective function to fitness,
(ii) create a probability distribution proportional to fitness, and
(iii) draw samples from this distribution.

The first three sections of this article discuss these steps. The final section discusses some results in the theory of proportional selection, including the schema theorem and the impact of the fitness function, and two characterizations of selective pressure.

C2.2.2 Fitness functions

The evaluation process of individuals in an evolutionary algorithm begins with the user-defined *objective function*,

$$f : A_x \rightarrow \mathbb{R}$$

where A_x is the object variable space. The objective function typically measures some cost to be minimized or some reward to be maximized. The definition of the objective function is, of course, application dependent. The characterization of how well evolutionary algorithms perform on different classes of objective functions is a topic of continuing research. However, a few general design principles are clear when using an evolutionary algorithm.

(i) The objective function must reflect the relevant measures to be optimized. Evolutionary algorithms are notoriously opportunistic, and there are several known instances of an algorithm optimizing the stated objective function, only to have the user realize that the objective function did not actually represent the intended measure.

(ii) The objective function should exhibit some regularities over the space defined by the selected representation.

(iii) The objective function should provide enough information to drive the selective pressure of the evolutionary algorithm. For example, 'needle-in-a-haystack' functions, i.e. functions that assign nearly equal value to every candidate solution except the optimum, should be avoided.

The *fitness function*

$$\Phi : A_x \to \mathbb{R}_+$$

maps the raw scores of the objective function to a non-negative interval. The fitness function is often a composition of the objective function and a scaling function g:

$$\Phi(a_i(t)) = g(f(a_i(t)))$$

where $a_i(t) \in A_x$. Such a mapping is necessary if the goal is to minimize the objective function, since higher fitness values correspond to lower objective values in this case. For example, one fitness function that might be used when the goal is to minimize the objective function is

$$\Phi(a_i(t)) = f_{\max} - f(a_i(t))$$

where $f_{\max}$ is the maximum value of the objective function. If the global maximum value of the objective function is unknown, an alternative is

$$\Phi(a_i(t)) = f_{\max}(t) - f(a_i(t))$$

where $f_{\max}(t)$ is the maximum observed value of the objective function up to time t. There are many other plausible alternatives, such as

$$\Phi(a_i(t)) = \frac{1}{1 + f(a_i(t)) - f_{\min}(t)}$$

where $f_{\min}(t)$ is the minimum observed value of the objective function up to time t. For maximization problems, this becomes

$$\Phi(a_i(t)) = \frac{1}{1 + f_{\max}(t) - f(a_i(t))}.$$

Note that the latter two fitness functions yield a range of (0, 1].

C2.2.2.1 Fitness scaling

As an evolutionary algorithm progresses, the population often becomes dominated by high-performance individuals with a narrow range of objective values. In this case, the fitness functions described above tend to assign similar fitness values to all members of the population, leading to a loss in the selective pressure toward the better individuals. To address this problem, *fitness scaling* methods that accentuate small differences in objective values are often used in order to maintain a productive level of selective pressure.

One approach to fitness scaling (Grefenstette 1986) is to define the fitness function as a time-varying linear transformation of the objective value, for example

$$\Phi(a_i(t)) = \alpha f(a_i(t)) - \beta(t)$$

where α is $+1$ for maximization problems and -1 for minimization problems, and $\beta(t)$ represents the worst value seen in the last few generations. Since $\beta(t)$ generally improves over time, this scaling method provides greater selection pressure later in the search. This method is sensitive, however, to 'lethals', poorly performing individuals that may occasionally arise through crossover or mutation. Smoother scaling can be achieved by defining $\beta(t)$ as a recency-weighted running average of the worst observed objective values, for example

$$\beta(t) = \delta\beta(t-1) + (1-\delta)(f_{\text{worst}}(t))$$

where δ is an update rate of, say, 0.1, and $f_{\text{worst}}(t)$ is the worst objective value in the population at time t.

Sigma scaling (Goldberg 1989) is based on the distribution of objective values within the current population. It is defined as follows:

$$\Phi(a_i(t)) = \begin{cases} f(a_i(t)) - (\bar{f}(t) - c\sigma_f(t)) & \text{if } f(a_i(t)) > (\bar{f}(t) - c\sigma_f(t)) \\ 0 & \text{otherwise} \end{cases}$$

where $\bar{f}(t)$ is the mean objective value of the current population, $\sigma_f(t)$ is the (sample) standard deviation of the objective values in the current population, and c is a constant, say $c = 2$. The idea is that $\bar{f}(t) - c\sigma_f(t)$ represents the least acceptable objective value for any reproducing individual. As the population improves, this statistic tracks the improvement, yielding a level of selective pressure that is sensitive to the spread of performance values in the population.

Fitness scaling methods based on power laws have also been proposed. A fixed transformation of the form

$$\Phi(a_i(t)) = f(a_i(t))^k,$$

where k is a problem-dependent parameter, is used by Gillies (1985). *Boltzmann selection* (de la Maza and Tidor 1993) is a power-law-based scaling method that draws upon techniques used in simulated annealing. The fitness function is a time-varying transformation given by

$$\Phi(a_i(t)) = \exp(f(a_i(t))/T)$$

where the parameter T can be used to control the level of selective pressure during the course of the evolution. It is suggested by de la Maza and Tidor (1993) that, if T decreases with time as in a simulated annealing procedure then a higher level of selective pressure results than with proportional selection without fitness scaling.

C2.2.3 Selection probabilities

Once the fitness values are assigned, the next step in proportional selection is to create a probability distribution such that the probability of selecting a given individual for reproduction is proportional to the individual's fitness. That is,

$$\mathrm{Pr}_{\mathrm{prop}}(i) = \frac{\Phi(i)}{\sum_{i=1}^{\mu} \Phi(i)}.$$

C2.2.4 Sampling

In an incremental, or steady-state, algorithm, the probability distribution can be used to select one parent at a time. This procedure is commonly called the *roulette wheel* sampling algorithm, since one can think of the probability distribution as defining a roulette wheel on which each slice has a width corresponding to the individual's selection probability, and the sampling can be envisioned as spinning the roulette wheel and testing which slice ends up at the top. The pseudocode for this is shown below:

Input: probability distribution Pr
Output: n, the selected parent

```
1   roulette wheel (Pr):
2       n ← 1;
3       sum ← Pr(n);
4       sample u ~ U(0, 1);
5       while sum < u do
                n ← (n + 1);
                sum ← sum + Pr(n);
        od
6       return (n);
```

In a generational algorithm, the entire population is replaced during each generation, so the probability distribution is sampled μ times. This could be implemented by μ independent calls to the roulette wheel procedure, but such an implementation may exhibit a high variance in the number of offspring assigned to each individual. For example, it is possible that the individual with the largest selection probability may be assigned no offspring in a particular generation. Baker (1987) developed an algorithm called *stochastic universal sampling* (SUS) that exhibits less variance than repeated calls to the roulette wheel algorithm. The idea is to make a single draw from a uniform distribution, and use this to determine how many offspring to assign to all parents. The pseudocode for SUS follows:

Input: a probability distribution, Pr; the total number of children to assign, λ.
Output: $\boldsymbol{c} = (c_1, \ldots, c_\mu)$, where c_i is the number of children assigned to individual a_i, and $\sum c_i = \lambda$.

```
1   SUS(Pr, λ):
2       sample u ~ U(0, 1/λ);
3       sum ← 0.0;
4       for i = 1 to μ do
5           c_i ← 0;
6           sum ← sum + Pr(i);
7           while u < sum do
8                   c_i ← c_i + 1;
9                   u ← u + 1/λ;
            od
        od
10      return c;
```

Note that the pseudocode allows for any number $\lambda > 0$ of children to be specified. If $\lambda = 1$, SUS behaves like the roulette wheel function. For generational algorithms, SUS is usually invoked with $\lambda = \mu$.

In can be shown that the expected number of offspring that SUS assigns to individual i is $\lambda \Pr(i)$, and that on each invocation of the procedure, SUS assigns either $\lfloor \lambda \Pr(i) \rfloor$ or $\lceil \lambda \Pr(i) \rceil$ offspring to individual i. Finally, SUS is optimally efficient, making a single pass over the individuals to assign all offspring.

C2.2.5 Theory

The section presents some results from the theory of proportional selection. First, the schema theorem is described, following by a discussion of the effects of the fitness function on the allocation of trials to schemata. The selective pressure of proportional selection is characterized in two ways. First, the selection differential describes the effects of selection on the mean population fitness. Second, the takeover time describes the convergence rate of population toward the optimal individual, in the absence of other genetic operators.

C2.2.5.1 The schema theorem

In the above description, $\Pr_{\text{prop}}(i)$ is the probability of selecting individual i for reproduction. In a generational evolutionary algorithm, the entire population is replaced, so the expected number of offspring of individual i is $\mu \Pr_{\text{prop}}(i)$. This value is called the *target sampling rate*, $\text{tsr}(a_i, t)$ of the individual (Grefenstette 1991). For any selection algorithm, the allocation of offspring to individuals induces a corresponding allocation to hyperplanes represented by the individuals:

$$\text{tsr}(H, t) =_{\text{def}} \sum_{i=1}^{m(H,t)} \frac{\text{tsr}(a_i, t)}{m(H, t)}$$

where $a_i \in H$ and $m(H, t)$ denotes the number of representatives of hyperplane H in population $P(t)$. In the remainder of this discussion, we will refer to $\text{tsr}(H, t)$ as the *target sampling rate* of H at time t.

For proportional selection, we have

$$\mathrm{tsr}(a_i, t) = \frac{\Phi(a_i)}{\bar{\Phi}(t)}$$

where Φ is the fitness function and $\bar{\Phi}(t)$ denotes the average fitness of the individuals in $P(t)$. The most important feature of proportional selection is that it induces the following target sampling rates for all hyperplanes in the population:

$$\begin{aligned}\mathrm{tsr}(H, t) &= \sum_{i=1}^{m(H,t)} \frac{\mathrm{tsr}(a_i, t)}{m(H, t)} \\ &= \sum_{i=1}^{m(H,t)} \frac{\Phi(a_i)}{\bar{\Phi}(t)\, m(H, t)} \\ &= \frac{\Phi(H, t)}{\bar{\Phi}(t)} \qquad \text{(C2.2.1)}\end{aligned}$$

where $\Phi(H, t)$ is simply the average fitness of the representatives of H in $P(t)$. This result is the heart of the schema theorem (Holland 1975), which has been called the *fundamental theorem of genetic algorithms* (Goldberg 1989).

Schema theorem. In a genetic algorithm using a proportional selection algorithm, the following holds for each hyperplane H represented in $P(t)$:

$$M(H, t+1) \geq M(H, t) \left(\frac{\Phi(H, t)}{\bar{\Phi}(t)} \right) (1 - p_{\mathrm{disr}}(H, t))$$

where $M(H, t)$ is the expected number of representatives of hyperplane H in $P(t)$, and $p_{\mathrm{disr}}(H, t)$ is the probability of disruption due to genetic operators such as crossover and mutation.

Holland provides an analysis of the disruptive effects of various genetic operators, and shows that hyperplanes with short defining lengths, for example, have a small chance of disruption due to one-point crossover and mutation operators. Others have extended this analysis to many varieties of genetic operators.

The main thrust of the schema theorem is that trials are allocated *in parallel* to a large number of hyperplanes (i.e. the ones with short definition lengths) according to the sampling rate (C2.2.1), with minor disruption from the recombination operators. Over succeeding generations, the number of trials allocated to extant short-definition-length hyperplanes with persistently above-average observed fitness is expected to grow rapidly, while trials to those with below-average observed fitness generally decline rapidly.

C2.2.5.2 Effects of the fitness function

In his early analysis of genetic algorithms, Holland implicitly assumes a nonnegative fitness and does not explicitly address the problem of mapping from the objective function to fitness in his brief discussion of function optimization (Holland 1975, ch 3). Consequently, many of the schema analysis results in the literature use the symbol f to refer to the *fitness* and not to *objective function* values. The methods mentioned above for mapping the objective function to the fitness values must be kept in mind when interpreting the schema theorem. For example, consider two genetic algorithms that both use proportional selection but that differ in that one uses the fitness function

$$\Phi_1(x) = \alpha f(x) + \beta$$

and the other uses the fitness function

$$\Phi_2(x) = \Phi_1(x) + \gamma$$

where $\gamma \neq 0$. Then for any hyperplane H represented in a given population $P(t)$, the target sampling rate for H in the first algorithm is

$$\mathrm{tsr}_1(H, t) = \frac{\Phi_1(H, t)}{\bar{\Phi}_1(t)}$$

while the target sampling rate for H in the second algorithm is

$$\begin{aligned}\mathrm{tsr}_2(H,t) &= \frac{\Phi_2(H,t)}{\bar{\Phi}_2(t)} \\ &= \frac{\Phi_1(H,t)+\gamma}{\bar{\Phi}_1(t)+\gamma}.\end{aligned}$$

Even though both genetic algorithms behave according to the schema theorem, they clearly allocate trials to hyperplane H at different rates, and thus produce entirely different sequences of populations. The relationship between the schema theorem and the objective function becomes even more complex if the fitness function Φ is dynamically scaled during the course of the algorithm. Clearly, the allocation of trials described by schema theorem depends on the precise form of the fitness function used in the evolutionary algorithm. And of course, crossover and mutation will also interact with selection.

C2.2.5.3 Selection differential

Drawing on the terminology of selective breeding, Mühlenbein and Schlierkamp-Voosen (1993) define the *selection differential* $S(t)$ of a selection method as the difference between the mean fitness of the selected parents and the mean fitness of the population at time t. For proportional selection, they show that the selection differential is given by

$$S(t) = \frac{\sigma_p^2(t)}{\bar{\Phi}(t)}$$

where $\sigma_p^2(t)$ is the fitness variance of the population at time t. From this formula, it is easy to see that, without dynamic fitness scaling, an evolutionary algorithm tends to stagnate over time since $\sigma_p^2(t)$ tends to decrease and $\bar{\Phi}(t)$ tends to increase. The fitness scaling techniques described above are intended to mitigate this effect. In addition, operators which produce random variation (e.g. mutation) can also be used to reduce stagnation in the population.

C2.2.5.4 Takeover time

Takeover time refers to the number of generations required for an evolutionary algorithm operating under selection alone (i.e. no other operators such as mutation or crossover) to converge to a population consisting entirely of instances of the optimal individual, starting from a population that contains a single instance of the optimal individual. Goldberg and Deb (1991) show that, assuming $\Phi = f$, the takeover time τ in a population of size μ for proportional selection is

$$\tau_1 = \frac{\mu \ln \mu - 1}{c}$$

for $f_1(x) = x^c$, and

$$\tau_2 = \frac{\mu \ln \mu}{c}$$

for $f_2(x) = \exp(cx)$. Goldberg and Deb compare these results with several other selection mechanisms and show that the takeover time for proportional selection (without fitness scaling) is larger than for many other selection methods.

References

Bäck T 1994 Selective pressure in evolutionary algorithms: a characterization of selection mechanisms *Proc. 1st IEEE Int. Conf. on Evolutionary Computation (Orlando, FL, June 1994)* (Piscataway, NJ: IEEE) pp 57–62

Baker J E 1987 Reducing bias and inefficiency in the selection algorithm *Proc. 2nd Int. Conf. on Genetic Algorithms (Cambridge, MA, 1987)* ed J Grefenstette (Hillsdale, NJ: Erlbaum) pp 14–21

de la Maza M and Tidor B 1993 An analysis of selection procedures with particular attention paid to proportional and Boltzmann selection *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 124–31

Gillies A M 1985 *Machine Learning Procedures for Generating Image Domain Feature Detectors* Doctoral Dissertation, University of Michigan, Ann Arbor

Goldberg D E 1989 *Genetic Algorithms in Search, Optimization, and Machine Learning* (Reading, MA: Addison-Wesley)

Goldberg D and Deb K 1991 A comparative analysis of selection schemes used in genetic algorithms *Foundations of Genetic Algorithms* ed G Rawlins (San Mateo, CA: Morgan Kaufmann) pp 69–93

Grefenstette J 1986 Optimization of control parameters for genetic algorithms *IEEE Trans. Syst. Man Cybernet.* **SMC-16** 122–8

——1991 Conditions for implicit parallelism *Foundations of Genetic Algorithms* ed G Rawlins (San Mateo, CA: Morgan Kaufmann) pp 252–61

Holland J H 1975 *Adaptation in Natural and Artificial Systems* (Ann Arbor, MI: University of Michigan Press)

Mühlenbein H and Schlierkamp-Voosen D 1993 Predictive models for the breeder genetic algorithm *Evolut. Comput.* **1** 25–49

C2.3 Tournament selection

Tobias Blickle

Abstract

This section is concerned with the description of tournament selection. An outline of the selection method is given and the basic algorithmic properties are summarized (time complexity and offspring variance). Furthermore, a mathematical analysis of the selection method is given that is based on fitness distributions. The analysis lays the ground for prediction of the expected number of offspring after selection as well as for derivation of the basic properties of a selection scheme, in particular takeover time, selection intensity, and loss of diversity.

C2.3.1 Working mechanism

In tournament selection a group of q individuals is randomly chosen from the population. They may be drawn from the population with or without replacement. This group takes part in a *tournament*; that is, a winning individual is determined depending on its fitness value. The best individual having the highest fitness value is usually chosen deterministically though occasionally a stochastic selection may be made. In both cases only the winner is inserted into the next population and the process is repeated λ times to obtain a new population. Often, tournaments are held between two individuals (binary tournament). However, this can be generalized to an arbitrary group size q called *tournament size*.

The following description assumes that the individuals are drawn with replacement and the winning individual is deterministically selected.

Input: Population $P(t) \in I^{\lambda}$, tournament size $q \in \{1, 2, \ldots, \lambda\}$
Output: Population after selection $P(t)'$
1 tournament(q, $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_\lambda$):
2 **for** $i \leftarrow 1$ **to** λ **do**
3 $\quad \boldsymbol{a}'_i \leftarrow$ best fit individual from q randomly chosen individuals from $\{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_\lambda\}$;
od
4 **return** $\{\boldsymbol{a}'_1, \ldots, \boldsymbol{a}'_\lambda\}$.

Tournament selection can be implemented very efficiently and has the time complexity $\mathcal{O}(\lambda)$ as no sorting of the population is required. However, the above algorithm leads to high variance in the expected number of offspring as λ independent trials are carried out.

Tournament selection is translation and scaling invariant (de la Maza and Tidor 1993). This means that a scaling or translation of the fitness value does not affect the behavior of the selection method. Therefore, scaling techniques as used for *proportional selection* are not necessary, simplifying the application of the selection method. C2.2

Furthermore, tournament selection is well suited for parallel evolutionary algorithms. In most selection schemes global calculations are necessary to compute the reproduction rates of the individuals. For example, in proportional selection the mean of the fitness values in the population is required, and in *ranking selection* and truncation selection a sorting of the whole population is necessary. However, in tournament selection the tournaments can be performed independently of each other such that only groups of q individuals need to communicate. C2.4

C2.3.2 Parameter settings

$q = 1$ corresponds to no selection at all (the individuals are randomly picked from the population). Binary tournament is equivalent to linear ranking selection with $\eta^- = 1/\lambda$ (Blickle and Thiele 1995a), where η^- gives the expected number of offspring of the worst individual. With increasing q the selection pressure increases (for a quantitative discussion of selection pressure see below). For many applications in genetic programming values $q \in \{6, \ldots, 10\}$ have been recommended.

C2.3.3 Formal description

Tournament selection has been well studied (Goldberg and Deb 1991, Bäck 1994, 1995, Blickle and Thiele 1995a, b, Miller and Goldberg 1995). The following description is based on the fitness distribution of the population.

Let $\gamma(P)$ denote the number of unique fitness values in the population. Then $\boldsymbol{\rho}(P) = (\rho_{F_1(P)}, \rho_{F_2(P)}, \ldots, \rho_{F_{\gamma(P)}(P)}) \in [0,1]^{\gamma(P)}$ is the fitness distribution of the population P, with $F_1(P) < F_2(P) < \cdots < F_{\gamma(P)}(P)$. $\rho_{F_i(P)}$ gives the proportion of individuals with fitness value $F_i(P)$ in the population P. Furthermore the cumulative fitness distribution is denoted by $\boldsymbol{R}(P) = (R_{F_1(P)}, R_{F_2(P)}, \ldots, R_{F_{\gamma(P)}(P)}) \in [0,1]^{\gamma(P)}$. $R_{F_i(P)}$ gives the number of individuals with fitness value $F_i(P)$ or less in the population P, i.e. $R_{F_i(P)} = \sum_{j=1}^{j=i} \rho_{F_j(P)}$ and $R_{F_0(P)} := 0$.

With these definitions, the selection operator s can be viewed as an operator on fitness distributions (Blickle and Thiele 1995b). The expected fitness distribution after tournament selection with tournament size q is $s_{\text{tour}}(q) : \mathbb{R}^{\gamma(P)} \mapsto \mathbb{R}^{\gamma(P)}$, $s_{\text{tour}}(q)(\boldsymbol{\rho}(P)) = (\rho'_{F_1(P)}, \rho'_{F_2(P)}, \ldots, \rho'_{F_\gamma(P)})$, where

$$\rho'_{F_i(P)} = (R_{F_i(P)})^q - (R_{F_{i-1}(P)})^q. \tag{C2.3.1}$$

The expected number of occurrences of an individual with fitness value $F_i(P)$ is given by $\rho'_{F_i(P)}/\rho_{F_i(P)}$. Consequently, stochastic universal sampling (Baker 1987) (see Section C2.2) can also be used for tournament selection. This almost completely reduces the usually high variance in the expected number of offspring. However, the time complexity of the selection algorithm increases to $\mathcal{O}(\lambda \ln \lambda)$ as calculation of the fitness distribution is required.

For the analytical analysis it is advantageous to use continuous fitness distributions. The continuous form of (C2.3.1) is given by

$$\bar{\rho}'(F) = q\bar{\rho}(F)\left(\bar{R}(F)\right)^{q-1} \tag{C2.3.2}$$

where $\bar{\rho}(F)$ is the continuous form of $\boldsymbol{\rho}(P)$ and $\bar{R}(F) = \int_{F_0(P)}^{F} \bar{\rho}(x)\,\mathrm{d}x$ is the cumulative continuous fitness distribution and $F_0(P) < F \leq F_{\gamma(P)}(P)$ the range of the distribution function $\bar{\rho}(F)$.

C2.3.4 Properties

C2.3.4.1 Concatenation of tournaments

An interesting property of tournament selection is the concatenation of several selection phases. Assuming an arbitrary population with a fitness distribution $\bar{\rho}$, tournament selection with tournament size q_1 is applied followed by tournament selection with tournament size q_2 on the resulting population and no recombination in between. The obtained expected fitness distribution is the same as if only a single tournament selection with tournament size $q_1 q_2$ were applied to the initial distribution $\bar{\rho}$ (Blickle and Thiele 1995b):

$$s_{\text{tour}}(q_2)(s_{\text{tour}}(q_1)(\bar{\rho})) = s_{\text{tour}}(q_1 q_2)(\bar{\rho}). \tag{C2.3.3}$$

C2.3.4.2 Takeover time

The takeover time was introduced by Goldberg and Deb (1991) to describe the selection pressure of a selection method. The takeover time τ^* is the number of generations needed under pure selection for a initial single best-fit individual to fill up the whole population. The takeover time can, for example, be calculated combining (C2.3.1) and (C2.3.3) as follows. Only the best individual is considered and its expected proportion ρ'_{best} after tournament selection can be obtained as $\rho'_{\text{best}} = 1 - (1 - 1/\lambda)^q$, which is a special case of (C2.3.1) using $\rho_{\text{best}} = 1/\lambda$ and $R_{\text{best}} = 1$. Performing τ such tournaments subsequently with

no recombination in between leads to $\hat{\rho}_{\text{best}} = 1 - (1 - 1/\lambda)^{q^\tau}$ by repeatedly applying (C2.3.3). Goldberg and Deb (1991) solved this equation for τ and gave the following approximation for the takeover time:

$$\tau^*_{\text{tour}}(q) \approx \frac{1}{\ln q}(\ln \lambda + \ln(\ln \lambda)). \tag{C2.3.4}$$

Figure C2.3.1 shows the dependence of the takeover time on the tournament size q. For scaling purposes an artificial population size of $\lambda = \text{e}$ is assumed, such that (C2.3.4) simplifies to $\tau^*_{\text{tour}}(q) \approx 1/\ln q$.

C2.3.4.3 Selection intensity

The selection intensity is another measure for the strength of selection which is borrowed from population genetics. The *selection intensity* S is the change in the average fitness of the population due to selection divided by the mean variance of the population before selection σ, that is, $S = (u^* - u)/\sigma$, with u average fitness before selection, and u^* average fitness after selection. To eliminate the dependence of the selection intensity on the initial distribution one usually assumes a Gaussian-distributed initial population (Mühlenbein and Schlierkamp-Voosen 1993). Under this assumption, the selection intensity of tournament selection is determined by

$$S_{\text{tour}}(q) = \int_{-\infty}^{\infty} qx \, \frac{1}{(2\pi)^{1/2}} \, \text{e}^{-x^2/2} \left(\int_{-\infty}^{x} \frac{1}{(2\pi)^{1/2}} \, \text{e}^{-y^2/2} \, \text{d}y \right)^{q-1} \, \text{d}x. \tag{C2.3.5}$$

The dependence of the selection intensity on the tournament size is shown in figure C2.3.1.

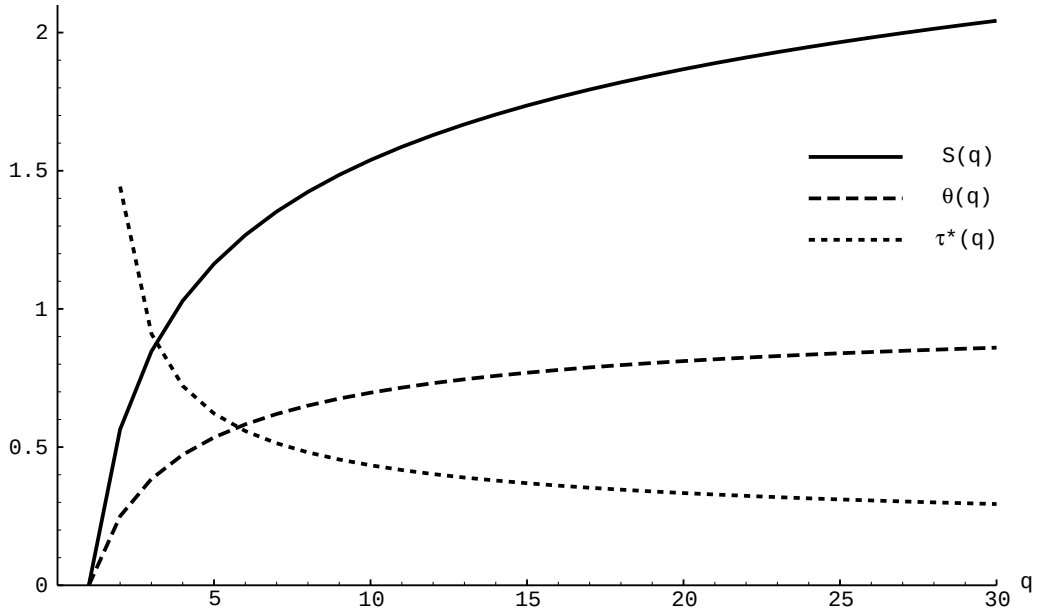


Figure C2.3.1. The selection intensity S, the loss of diversity θ, and the takeover time τ^* (for $\lambda = \text{e}$) of tournament selection in dependence on the tournament size q.

The known exact solutions of the integral equation (C2.3.5) are given in table C2.3.1. These values can also be obtained using the results of the order statistics theory (Bäck 1995). The following formula was derived by Blickle and Thiele (1995b) and approximates the selection intensity with a relative error of less than 1% for tournament sizes of $q > 5$:

$$S_{\text{tour}}(q) \approx (2(\ln(q) - \ln((4.14 \ \ln(q))^{1/2})))^{1/2}.$$

C2.3.4.4 Loss of diversity

During every selection phase bad individuals are replaced by copies of better ones. Thereby a certain amount of 'genetic material' contained in the bad individuals is lost. The *loss of diversity* θ is the

Table C2.3.1. Known exact values for the selection intensity of tournament selection.

q	1	2	3	4	5
$S_{\text{tour}}(q)$	0	$\frac{1}{\pi^{1/2}}$	$\frac{3}{2\pi^{1/2}}$	$\frac{6}{\pi\pi^{1/2}}\tan^{-1}2^{1/2}$	$\frac{10}{\pi^{1/2}}\left(\frac{3}{2\pi}\tan^{-1}2^{1/2}-\frac{1}{4}\right)$

proportion of the population that is not selected for the next population (Blickle and Thiele 1995b). Baker (1989) introduces a similar measure called 'reproduction rate, RR'. RR gives the percentage of individuals that is selected to reproduce, hence $\text{RR} = 100(1-\theta)$.

For tournament selection this value computes to (Blickle and Thiele 1995b)

$$\theta_{\text{tour}}(q) = q^{-1/(q-1)} - q^{-q/(q-1)}.$$

It is interesting to note that the loss of diversity is independent of the initial fitness distribution $\bar{\rho}$. Furthermore, a relatively moderate tournament size of $q = 5$ leads to a loss of diversity of almost 50% (see figure C2.3.1).

References

Bäck T 1994 Selective pressure in evolutionary algorithms: a characterization of selection mechanisms *Proc. 1st IEEE Conf. on Evolutionary Computation (Orlando, FL, June 1994)* (Piscataway, NJ: IEEE) pp 57–62

——1995 Generalized convergence models for tournament- and (μ, λ)-selection *Proc. 6th Int. Conf. on Genetic Algorithms (Pittsburg, PA, July 1995)* ed L J Eshelman (San Mateo, CA: Morgan Kaufmann) pp 2–8

Baker J E 1987 Reducing bias and inefficiency in the selection algorithm *Proc. 2nd Int. Conf. on Genetic Algorithms (Cambridge, MA, 1987)* ed J J Grefenstette (Hillsdale, NJ: Erlbaum) pp 14–21

——1989 *An Analysis of the Effects of Selection in Genetic Algorithms* PhD Thesis, Graduate School of Vanderbilt University, Nashville, TN

Blickle T and Thiele L 1995a *A Comparison of Selection Schemes used in Genetic Algorithms* Technical Report 11, Computer Engineering and Communication Networks Lab (TIK), Swiss Federal Institute of Technology (ETH) Zurich

——1995b A mathematical analysis of tournament selection *Proc. 6th Int. Conf. on Genetic Algorithms (Pittsburg, PA, July 1995)* ed L J Eshelman (San Mateo, CA: Morgan Kaufmann) pp 9–16

de la Maza M and Tidor B 1993 An analysis of selection procedures with particular attention paid to proportional and Boltzmann selection *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 124–31

Goldberg D E and Deb K 1991 A comparative analysis of selection schemes used in genetic algorithms *Foundations of Genetic Algorithms* ed G Rawlins (San Mateo, CA: Morgan Kaufmann) pp 69–93

Miller B L and Goldberg D E 1995 *Genetic Algorithms, Tournament Selection, and the Effects of Noise* Technical Report 95006, Illinois Genetic Algorithm Laboratory, University of Urbana-Champaign

Mühlenbein H and Schlierkamp-Voosen D 1993 Predictive models for the breeder genetic algorithm *Evolut. Comput.* **1** 25–49

C2.4 Rank-based selection

John Grefenstette

Abstract

Rank-based selection assigns a reproductive or survival probability to each individual that depends only on the rank ordering of the individuals in the current population. The section presents a brief discussion of ranking, including linear, nonlinear, (μ, λ), and $(\mu + \lambda)$ methods. The theory of rank-based selection is briefly outlined, including a discussion of implicit parallelism and characterizations of selective pressure in rank-based evolutionary algorithms.

C2.4.1 Introduction

Selection is the process of choosing individuals for reproduction or survival in an evolutionary algorithm. *Rank-based selection* or *ranking* means that only the rank ordering of the fitness of the individuals within the current population determines the probability of selection.

As discussed in Section C2.2, the selection process may be decomposed into distinct steps: C2.2

(i) Map the objective function to fitness.
(ii) Create a probability distribution based on fitness.
(iii) Draw samples from this distribution.

Ranking simplifies step (i), the mapping from the objective function f to the fitness function Φ. All that is needed is

$$\Phi(a_i) = \delta f(a_i)$$

where δ is $+1$ for maximization problems and -1 for minimization problems.

Ranking also eliminates the need for *fitness scaling*, since selection pressure is maintained even if the objective function values within the population converge to a very narrow range, as often happens as the population evolves. C2.2.2.1

This section discusses step (ii), the creation of the selection probability distribution based on fitness. The final step (iii) is independent of the selection method, and the *stochastic universal sampling* algorithm is an appropriate sampling procedure. C2.2.4

Besides its simplicity, other motivations for using rank-based selection include:

(i) Under proportional selection, a 'super' individual, i.e. an individual with vastly superior objective value, might completely take over the population in a single generation unless an artificial limit is placed on the maximum number of offspring for any individual. Ranking helps prevent premature convergence due to 'super' individuals, since the best individual is always assigned the same selection probability, regardless of its objective value.
(ii) Ranking may be a natural choice for problems in which it is difficult to precisely specify an objective function, e.g. if the objective function involves a person's subjective preference for alternative solutions. For such problems it may make little sense to pay too much attention to the exact values of the objective function, if exact values exist at all.

The following sections describe various forms of linear and nonlinear ranking algorithms. The final section presents some of the theory of rank-based selection.

C2.4.2 Linear ranking

Linear ranking assigns a selection probability to each individual that is proportional to the individual's rank (where the rank of the least fit is defined to be zero and the rank of the most fit is defined to be $\mu - 1$, given a population of size μ). For a generational algorithm, linear ranking can be implemented by specifying a single parameter, β_{rank}, the expected number of offspring to be allocated to the best individual during each generation. The selection probability for individual i is then defined as follows:

$$\Pr_{\text{lin\_rank}}(i) = \frac{\alpha_{\text{rank}} + [\text{rank}(i)/(\mu - 1)](\beta_{\text{rank}} - \alpha_{\text{rank}})}{\mu}$$

where α_{rank} is the number of offspring allocated to the worst individual. The sum of the selection probabilities is then

$$\begin{aligned}\sum_{i=0}^{\mu-1} \frac{\alpha_{\text{rank}} + [\text{rank(i)}/(\mu - 1)](\beta_{\text{rank}} - \alpha_{\text{rank}})}{\mu} &= \alpha_{\text{rank}} + \frac{\beta_{\text{rank}} - \alpha_{\text{rank}}}{\mu(\mu - 1)} \sum_{i=0}^{\mu-1} i \\ &= \alpha_{\text{rank}} + \tfrac{1}{2}(\beta_{\text{rank}} - \alpha_{\text{rank}}) \\ &= \tfrac{1}{2}(\beta_{\text{rank}} + \alpha_{\text{rank}}).\end{aligned}$$

It follows that $\alpha_{\text{rank}} = 2 - \beta_{\text{rank}}$, and $1 \leq \beta_{\text{rank}} \leq 2$. That is, the expected number of offspring of the best individual is no more than twice that of the population average. This shows how ranking can avoid premature convergence caused by 'super' individuals.

C2.4.3 Nonlinear ranking

Nonlinear ranking assigns selection probabilities that are based on each individual's rank, but are not proportional to the rank. For example, the selection probabilities might be proportional to the square of the rank:

$$\Pr_{\text{sq\_rank}}(i) = \frac{\alpha + [\text{rank}(i)^2/(\mu - 1)^2](\beta - \alpha)}{c}$$

where $c = (\beta - \alpha)\mu(2\mu - 1)/6(\mu - 1) + \mu\alpha$ is a normalization factor. This version has two parameters, α and β, where $0 < \alpha < \beta$, such that the selection probabilities range from α/c to β/c.

Even more aggressive forms of ranking are possible. For example, one could assign selection probabilities based on a geometric distribution:

$$\Pr_{\text{geom\_rank}} = \alpha(1 - \alpha)^{\mu - 1 - \text{rank}(i)}.$$

This distribution arises if selection occurs as a result of independent Bernoulli trials over the individuals in rank order, with the probability of selecting the next individual equal to α, and was introduced in the GENITOR system (Whitley and Kauth 1988, Whitley 1989).

Another variation that provides exponential probabilities based on rank is

$$\Pr_{\text{exp\_rank}}(i) = \frac{1 - \text{e}^{-\text{rank}(i)}}{c} \tag{C2.4.1}$$

for a suitable normalization factor c. Both of the latter methods strongly bias the selection toward the best few individuals in the population, perhaps at the cost of premature convergence.

C2.4.4 (μ, λ), $(\mu + \lambda)$ and threshold selection

The (μ, λ) and $(\mu + \lambda)$ methods used in *evolution strategies* (Schwefel 1977) are deterministic rank-based selection methods. In (μ, λ) selection, $\lambda = k\mu$ for some $k > 1$. The process is that k offspring are generated from each parent in the current population through mutation or possibly recombination, and the best μ offspring are selected for retention. This method is similar to the technique called *beam search* in artificial intelligence (Shapiro 1990). Experimental studies indicate that a value of $k \approx 7$ is optimal (Schwefel 1987). B1.3

In $(\mu + \lambda)$ selection, the best μ individuals are selected from the union of the μ parents and the λ offspring. Thus, $(\mu + \lambda)$ is an elitist method, since it always retains the best individuals unless they are

replaced by superior individuals. According to Bäck and Schwefel (1993), the (μ, λ) method is preferable to $(\mu + \lambda)$, since it is more robust on probabilistic or changing environments.

The (μ, λ) method is closely related to methods known as *threshold selection* or *truncation selection* in the genetic algorithm literature. In threshold selection the best $T\mu$ individuals are assigned a uniform selection probability, and the rest of the population is discarded:

$$\Pr_{\text{thresh\_rank}}(i) = \begin{cases} 0 & \text{if rank}(i) < (1-T)\mu \\ 1/T\mu & \text{otherwise.} \end{cases}$$

The parameter T is the called the *threshold*, where $0 < T \leq 1$. According to Mühlenbein and Schlierkamp-Voosen (1993), T should be chosen in the range 0.1–0.5.

Threshold selection is essentially a (μ', λ) method, with $\mu' = T\mu$ and $\lambda = \mu$, except that threshold selection is usually implemented as a probabilistic procedure using the distribution $\Pr_{\text{thresh\_rank}}$, while systems using (μ, λ) are usually deterministic.

C2.4.5 Theory

The theory of rank-based selection has received less attention than the proportional selection method, due in part to the difficulties in applying the schema theorem to ranking. The next subsection describes the issues that arise in the schema analysis of ranking, and shows that ranking does exhibit a form of implicit parallelism. Characterizations of the selective pressure of ranking are also described, including its fertility rate, selective differential, and takeover time. Finally, a simple substitution result is mentioned.

C2.4.5.1 Ranking and implicit parallelism

The use of rank-based selection makes it difficult to relate the schema theorem to the original objective function, since the mean observed rank of a schema is generally unrelated to the mean observed objective value for that schema. As a result, the relative *target sampling rates* of two schemata under ranking cannot be predicted based on the mean objective values of the schemata, in contrast to proportional selection. For example, consider the following case: C2.2.5.1

$$f(a_1) = 59 \qquad f(a_2) = 15 \qquad f(a_3) = 5 \qquad f(a_4) = 1 \qquad f(a_5) = 0$$

where

$$a_1, a_4, a_5 \in H_1 \qquad a_2, a_3 \in H_2.$$

Assume that the goal is to maximize the objective function f. Even though $\overline{f}(H_1) = 20 > 10 = \overline{f}(H_2)$, ranking will assign a higher target sampling rate to H_2 than to H_1.

However, ranking does exhibit a weaker form of *implicit parallelism*, meaning that it allocates search effort in a way that differentiates among a large number of competing areas of the search space on the basis of a limited number of explicit evaluations of knowledge structures (Grefenstette 1991). The following definitions assume that the goal is to maximize the objective function.

A fitness function Φ is called *monotonic* if

$$\Phi(a_i) \leq \Phi(a_j) \Leftrightarrow f(a_i) \leq f(a_j).$$

That is, a monotonic fitness function does not reverse the sense of any pairwise ranking provided by the objective function. A fitness function is called *strictly monotonic* if it is monotonic and

$$f(a_i) < f(a_j) \Rightarrow \Phi(a_i) < \Phi(a_j).$$

A strictly monotonic fitness function preserves the relative ranking of any two individuals in the search space with distinct objective function values. Since $\Phi(a_i) = \delta f(a_i)$, ranking uses a strictly monotonic fitness function by definition.

Likewise, a selection algorithm is called *monotonic* if

$$\text{tsr}(a_i) \leq \text{tsr}(a_j) \Leftrightarrow \Phi(a_i) \leq \Phi(a_j)$$

where tsr(a) is the target sampling rate, or expected number of offspring, for individual a. That is, a monotonic selection algorithm is one that respects the *survival-of-the-fittest* principle. A selection algorithm is called *strictly monotonic* if it is monotonic and

$$\Phi(a_i) < \Phi(a_j) \Rightarrow \text{tsr}(a_i) < \text{tsr}(a_j).$$

A strictly monotonic selection algorithm assigns a higher selection probability to individuals with better fitness values. Linear ranking selection and proportional selection are both strictly monotonic, whereas threshold selection is monotonic but not strict, since it may assign the same number of offspring to individuals with different fitness values.

Finally, an evolutionary algorithm is called *admissible* if its fitness function and selection algorithm are both monotonic. An evolutionary algorithm is *strict* iff its fitness function and selection algorithm are both strictly monotonic.

Now, consider two arbitrary subsets of the solution space, A and B, sorted by objective function value. By definition, B *partially dominates* A ($A \prec B$) at time t if each representative of B is at least as good as the corresponding representative of A. The following theorem (Grefenstette 1991) partially characterizes the implicit parallelism exhibited by ranking (any many other selection methods):

Implicit parallelism of admissible evolutionary algorithms. In any admissible evolutionary algorithm, if ($A \prec B$) then tsr(A) $\leq$ tsr(B). Furthermore, in any strict evolutionary algorithm, if ($A \prec B$) then tsr(A) $<$ tsr(B).

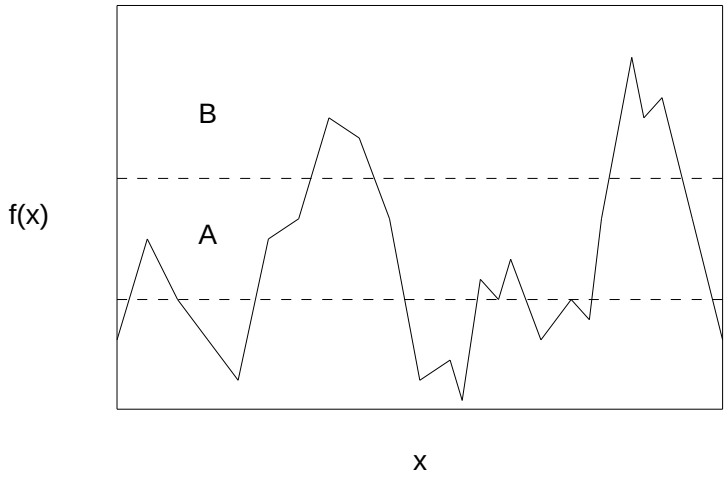


Figure C2.4.1. Two regions defined by range of objective values.

One illustration of this result to rank-based selection is shown in figure C2.4.1. Let A be the set of points in the space with objective function values between the dotted lines. Let B be the set of points in the space with objective values above the region between the dotted lines. Then, in any population that contains points from both set A and set B, the number of offspring allocated to B by any strict evolutionary algorithm grows strictly faster than the number allocated to set A, since any subset of B dominates any subset of A. This example illustrates *implicit parallelism* because it holds no matter where the dotted lines are drawn. This result holds not only for rank-based selection, but for any fitness function and selection algorithm that satisfy the requirement of admissibility.

C2.4.5.2 Fertility rate

The *fertility rate* $\mathcal{F}$ of a selection method is the proportion of the population that is expected to have at least one offspring as a result of the selection process. Other terms that have been used for this include *fertility factor* (Baker 1985, 1987), *reproductive rate* (Baker, 1989), and *diversity* (Blickle and Thiele, 1995).

Baker (1987, 1989) shows that, for linear ranking, the fertility rate obeys the following formula:

$$\mathcal{F} = 1 - \frac{\beta - 1}{4}$$

where β is the number of offspring allocated to the best individual, $1 \leq \beta \leq 2$. So $\mathcal{F}$ ranges in value from 1 (if $\beta = 1$) to 0.75 (if $\beta = 2$) for linear ranking.

C2.4.5.3 Selection differential

Drawing on the terminology of selective breeding, Mühlenbein and Schlierkamp-Voosen (1993) define the *selection differential* $S(t)$ of a selection method as the difference between the mean fitness of the selected parents and the mean fitness of the population at time t. If the fitness values are normally distributed the selection differential for truncation selection is approximately

$$S(t) \approx I\sigma_{\mathrm{p}}$$

where σ_{p} is the standard deviation of the fitness values in the population, and I is a value called the *selection intensity*. Bäck (1995) quantifies the selection intensity for general (μ, λ) selection as follows:

$$I = \frac{1}{\mu} \sum_{i=\lambda-\mu+1}^{\lambda} \boldsymbol{E}(Z_{i:\lambda})$$

where $Z_{i:\lambda}$ are order statistics based in the fitness of individuals in the current population. That is, I is the average of the expectations of the μ best samples taken from iid normally distributed random variables Z. This analysis shows that I is approximately proportional to λ/μ, and experimental studies confirm this relationship (Bäck 1995, Mühlenbein and Schlierkamp-Voosen 1993).

C2.4.5.4 Takeover time

Takeover time refers to the number of generations required for an evolutionary algorithm operating under selection alone (i.e. no other operators such as mutation or crossover) to converge to a population consisting entirely of instances of the optimal individual, starting from a population that contains a single instance of the optimal individual. According to Goldberg and Deb (1991), the approximate takeover time τ in a population of size μ for rank-based selection is

$$\tau \approx \frac{\ln \mu + \ln(\ln \mu)}{\ln 2}$$

for linear ranking with $\beta_{\mathrm{rank}} = 2$ and

$$\tau \approx \frac{2}{\mu - 1} \ln(\mu - 1)$$

for linear ranking with $1 < \beta_{\mathrm{rank}} < 2$.

C2.4.5.5 Substitution theorem

One interesting feature of rank-based selection is that it is clearly less sensitive to the objective function than proportional selection. As a result, it possible to make the following observation about evolutionary algorithms that use rank-based selection:

Substitution theorem. Let EA be an evolutionary algorithm that uses rank-based selection, along with any forms of mutation and recombination that are independent of the the objective values of individuals. If EA optimizes an objective function f then EA also optimizes the function $g \circ f$, for any monotonically increasing g.

Proof. For any monotonically increasing function g, the composition $g \circ f$ induces the same rank ordering of the search space as f. It follows that a rank-based algorithm EA produces an identical sequence of populations for objective functions f and $g \circ f$, assuming that mutation and recombination in EA are independent of the the objective values of individuals. Since f and $g \circ f$ have the same optimal solutions, the result follows.

For example, a rank-based evolutionary algorithm that optimizes a given function $f(x)$ in t steps will also optimize the function $(f(x))^n$ in t steps, for any even $n > 0$.

References

Bäck T 1995 Generalized convergence models for tournament- and (μ, λ)-selection *Proc. 6th Int. Conf. on Genetic Algorithms (Pittsburgh, PA, July 1995)* ed L J Eshelman (San Mateo, CA: Morgan Kaufmann) pp 2–8

Bäck T and H-P Schwefel 1993 An overview of evolutionary algorithms for parameter optimization *Evolut. Comput.* **1** 1–23

Baker J 1985 Adaptive selection methods for genetic algorithms *Proc. 1st Int. Conf. on Genetic Algorithms (Pittsburgh, PA, July 1985)* ed J J Grefenstette (Hillsdale, NJ: Lawrence Erlbaum) pp 101–11

——1987 Reducing bias and inefficiency in the selection algorithm *Proc. 2nd Int. Conf. on Genetic Algorithms (Cambridge, MA, 1987)* ed J J Grefenstette (Hillsdale, NJ: Erlbaum) pp 14–21

——1989 *Analysis of the Effects of Selection in Genetic Algorithms* Doctoral Dissertation, Department of Computer Science, Vanderbilt University

Blickle T and Thiele L 1995 A mathematical analysis of tournament selection *Proc. 6th Int. Conf. on Genetic Algorithms (Pittsburgh, PA, July 1995)* ed L Eshelman (San Mateo, CA: Morgan Kaufmann) pp 9–16

Goldberg D and Deb K 1991 A comparative analysis of selection schemes used in genetic algorithms *Foundations of Genetic Algorithms* ed G Rawlins (San Mateo, CA: Morgan Kaufmann) pp 69–93

Grefenstette J 1991 Conditions for implicit parallelism *Foundations of Genetic Algorithms* ed G Rawlins (San Mateo, CA: Morgan Kaufmann) pp 252–61

Mühlenbein H and Schlierkamp-Voosen D 1993 Predictive models for the breeder genetic algorithm *Evolut. Comput.* **1** 25–49

Schwefel H-P 1977 *Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie* (*Interdisciplinary System Research 26*) (Basel: Birkhäuser)

——1987 Collective phenomena in evolutionary systems *Preprints of the 31st Ann. Meeting International Society for General Systems Research (Budapest)* vol 2, pp 1025–33

Shapiro S C (ed) 1990 *Encyclopedia of Artificial Intelligence* vol 1 (New York: Wiley)

Whitley D 1989 The GENITOR algorithm and selective pressure: why rank-based allocation of reproductive trials is best *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, June 1989)* ed J Schaffer (San Mateo, CA: Morgan Kaufmann) pp 116–21

Whitley D and Kauth J 1988 GENITOR: a different genetic algorithm *Proc. Rocky Mountain Conf. on Artificial Intelligence (Denver, CO)* pp 118–30

C2.5 Boltzmann selection

Samir W Mahfoud

Abstract

Boltzmann evolutionary algorithms and their embedded selection mechanisms are traditionally employed to prolong search. After a brief introduction, a precursor called simulated annealing is outlined. A prominent type of Boltzmann evolutionary algorithm called parallel recombinative simulated annealing is then covered in depth. A proof of global convergence for this type of algorithm is illustrated.

C2.5.1 Introduction

Boltzmann selection mechanisms thermodynamically control the selection pressure in an evolutionary algorithm (EA), using principles from *simulated annealing* (SA) (Kirpatrick *et al* 1983). Boltzmann selection mechanisms can be used to indefinitely prolong an EA's search, in order to locate better final solutions.

In EAs that employ Boltzmann selection mechanisms, it is often impossible to separate the selection mechanism from the rest of the EA. In fact, the mechanics of the recombination and neighborhood operators are critical to the generation of the proper temporal population distributions. Therefore, most of the following discusses *Boltzmann EAs* rather than Boltzmann selection mechanisms in isolation.

Boltzmann EAs represent parallel extensions of the inherently serial SA. In addition, theoretical proofs of asymptotic, global convergence for SA carry over to certain Boltzmann selection EAs (Mahfoud and Goldberg 1995).

The heart of Boltzmann selection mechanisms is the *Boltzmann trial*, a competition between current solution i and alternative solution j, in which i wins with logistic probability

$$\frac{1}{1+\mathrm{e}^{(f_i-f_j)/T}} \tag{C2.5.1}$$

where T is temperature and f_i is the energy, cost, or objective function value (assuming minimization) of solution i. Slight variations of the Boltzmann trial exist, but all variations essentially accomplish the same thing when iterated (the winner of a trial becomes solution i for the next trial): at fixed T, given a sufficient number of Boltzmann trials, a Boltzmann distribution arises among the winning solutions (over time). The intent of the Boltzmann trial is that at high T, i and j win with nearly equal probabilities, making the system fluctuate wildly from solution to solution; at low T, the better of the two solutions nearly always wins, resulting in a relatively stable system.

Several types of Boltzmann algorithm exist, each designed for slightly different purposes. *Boltzmann tournament selection* (Goldberg 1990, Mahfoud 1993) is designed to give the population *niching* capabilities (Mahfoud 1995), but is not able to significantly slow the population's convergence. (*Convergence* refers to a population's decrease in diversity over time, as measured by an appropriate diversity measure.) Whether any Boltzmann EA is capable of performing effective niching remains an open question. C2.3

The Boltzmann selection method of de la Maza and Tidor (1993) scales the fitnesses of population elements, following fitness assignment, according to the Boltzmann distribution. It is designed to control the convergence of traditional selection.

Parallel recombinative simulated annealing (PRSA) (Mahfoud and Goldberg 1992, 1995) allows control of EA convergence, achieves a true parallelization of SA, and inherits SA's convergence proofs. PRSA is the Boltzmann EA discussed in the remainder of this section.

C2.5.2 Simulated annealing

SA is an optimization technique, analogous to the physical process of annealing. SA starts with a high temperature T and any initial state. A neighborhood operator is applied to the current state i to yield state j. If $f_j < f_i$, j becomes the current state. Otherwise j becomes the current state with probability $e^{(f_i - f_j)/T}$. (If j does not become the current state, i remains the current state.) The application of the neighborhood operator and the probabilistic acceptance of the newly generated state are repeated either for a fixed number of iterations or until a quasi-equilibrium is reached. The entire above-described procedure is performed repeatedly, each time starting from the current i and from a lower T.

At any given T, a sufficient number of iterations always leads to equilibrium, at which point the temporal distribution of accepted states is stationary. (This stationary distribution is Boltzmann.) The SA algorithm, as described above, is called the *Metropolis algorithm*. What distinguishes the Metropolis algorithm is the criterion by which the newly generated state is accepted or rejected. An alternative criterion is that of equation (C2.5.1). Both criteria lead to a Boltzmann distribution.

The key to achieving good performance with SA, as well as to proving global convergence, is that a stationary distribution must be reached at each temperature, and cooling (lowering T) must proceed sufficiently slowly.

C2.5.3 Working mechanism for parallel recombinative simulated annealing

PRSA is a population-level implementation of simulated annealing. Instead of processing one solution at a time, it processes an entire population of solutions in parallel, using a recombination operator (typically *crossover*) and a neighborhood operator (typically *mutation*). The combination of crossover and mutation produces a population-level neighborhood operator whose action on the entire population parallels the action of SA's neighborhood operator on a single solution. (See figure C2.5.1.) It is interesting to note that without crossover, PRSA would be equivalent to running μ independent SAs, where μ is population size. Without mutation, PRSA's global convergence proofs would no longer hold.

C3.3, C3.2

PRSA works by pairing all population elements, at random, for crossover each generation. After crossover and mutation, children compete against their parents in Boltzmann trials. Winners advance to the next generation.

In the Boltzmann trial step, many competitions are possible between two children and two parents. One possibility, *double acceptance/rejection*, allows both parents to compete as a unit against both children: the sum of the two parents' energies should be substituted for f_i in equation (C2.5.1); the sum of the two childrens' energies, for f_j. A second possibility, *single acceptance/rejection*, holds two competitions, each time pitting one child against one parent. There are several possible single acceptance/rejection competitions. For instance, each parent can always compete against the child formed from its own right end and the other parent's left end (assuming single-point crossover). Other possibilities and their consequences are outlined by Mahfoud and Goldberg (1995).

C2.5.4 Pseudocode for a common variation of parallel recombinative simulated annealing

The pseudocode at the top of the next page describes a common variation of PRSA that employs single acceptance/rejection competitions, a static stopping criterion, and random—without replacement—pairing of population elements for recombination. The cooling schedule is set by the two functions, initialize_temperature() and adjust_temperature(). These two functions, as well as initialize_population(), are shown without arguments, because their arguments depend upon the type of cooling schedule and initialization chosen by the user. The function random() simply returns a pseudorandom real number on the interval $(0, 1)$.

C2.5.5 Parameters and their settings

PRSA allows the use of any recombination and neighborhood operators. It performs minimization by default; maximization can be accomplished by reversing the sign of all objective function values. Population size (μ) remains constant from generation to generation. The number of generations the algorithm runs can either be fixed, as in the pseudocode, or dynamic, determined by a user-specified stopping or convergence criterion that is perhaps tied to the cooling schedule.

Input: g—number of generations to run, μ—population size
Output: $P(g)$—the final population

$P(0) \leftarrow$ initialize_population()
$T(1) \leftarrow$ initialize_temperature()
for $t \leftarrow 1$ **to** g **do**
  $P(t) \leftarrow$ shuffle($P(t-1)$)
  for $i \leftarrow 0$ **to** $\mu/2 - 1$ **do**
    $p_1 \leftarrow \boldsymbol{a}_{2i+1}(t)$
    $p_2 \leftarrow \boldsymbol{a}_{2i+2}(t)$
    $\{c_1, c_2\} \leftarrow$ recombine(p_1, p_2)
    $c_1' \leftarrow$ neighborhood(c_1)
    $c_2' \leftarrow$ neighborhood(c_2)
    if random() $> [1 + \mathrm{e}^{[f(p_1) - f(c_1')]/T(t)}]^{-1}$ **then** $\boldsymbol{a}_{2i+1}(t) \leftarrow c_1'$ **fi**
    if random() $> [1 + \mathrm{e}^{[f(p_2) - f(c_2')]/T(t)}]^{-1}$ **then** $\boldsymbol{a}_{2i+2}(t) \leftarrow c_2'$ **fi**
  od
  $T(t+1) \leftarrow$ adjust_temperature()
od

PRSA requires a user to select a population size, a type of competition, recombination and neighborhood operators, and a cooling schedule. Prior research offers some guidelines (Mahfoud and Goldberg 1992, 1995). A good rule of thumb for *population size* is to choose as large a population size as system limitations and time constraints allow. In general, smaller populations require longer cooling schedules. The type of competition previously employed is single acceptance/rejection, in which each parent competes against the child formed from its own right end and the other parent's left end (under single-point crossover). E1.1

Appropriate recombination and neighborhood operators are problem specific. For example, in optimization of traditional binary encodings, one might employ single-point crossover and mutation; in permutation problems, permutation-based crossover and inversion would be more appropriate.

Many styles of cooling schedule exist, but their discussion is beyond the scope of this section. Several studies contain thorough discussions of cooling (Aarts and Korst 1989, Azencott 1992, Ingber and Rosen 1992, Romeo and Sangiovanni-Vincentelli 1991). Perhaps the simplest type of cooling schedule is to start at a high T, and to periodically lower T through multiplication by a positive constant such as 0.95. At each T, a number of generations are performed. In general, the more generations performed at each T and the higher the multiplicative constant, the better the end result.

C2.5.6 Global convergence theory and proofs

The most straightforward global convergence proof for any variation of PRSA shows that the variation is a special case of standard SA. This results in the transfer of SA's convergence proof to the PRSA variant. Details of PRSA's convergence proofs are given by Mahfoud and Goldberg (1995).

The variation of PRSA that we consider employs selection of parents with replacement, and double acceptance/rejection. No population element may be selected as both parents. (Self-mating is disallowed.)

Many authors have taken the viewpoint that SA is essentially an EA with a population size of one. Our proof takes the opposite viewpoint, showing an EA (PRSA) to be a special case of SA. To see this, concatenate all strings of the PRSA population in a side-by-side fashion to form one superstring. Define the fitness of this superstring to be the sum of the individual fitnesses of its component substrings (the former population elements). Let cost be the negated fitness of this superstring. The cost function will reach a global minimum only when each substring is identically at a global maximum. Thus, to maximize all elements of the former population, PRSA can search for a global minimum for the cost function assigned to its superstring.

Consider the superstring as our structure to be optimized. Our chosen variation of PRSA, as displayed graphically in figure C2.5.1, is now a special case of SA, in which the crossover-plus-mutation neighborhood operator is applied to selected portions of the superstring to generate new superstrings.

Crossover-plus-mutation's net effect as a population-level neighborhood operator is to swap two blocks of the superstring, and then probabilistically flip bits of these swapped blocks and of two other blocks (the other halves of each parent).

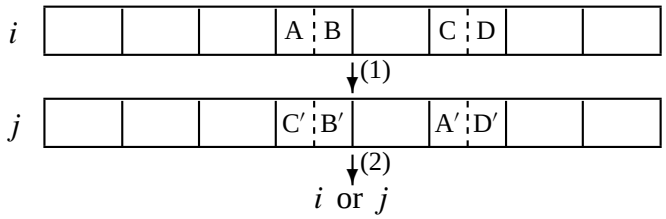


Figure C2.5.1. The population, after application of crossover and mutation (step 1), transitions from superstring i to superstring j. After a Boltzmann trial (step 2), either i or j becomes the current population. Individual population elements are represented as rectangles within the superstrings. Blocks A, B, C, and D represent portions of individual population elements, prior to crossover and mutation. Crossover points are shown as dashed lines. Blocks A′, B′, C′, and D′ result from applying mutation to A, B, C, and D.

As a special case of SA, the chosen variation of PRSA inherits the global convergence proof of SA, provided the population-level neighborhood operator meets certain conditions. According to Aarts and Korst (1989), two conditions on the neighborhood generation mechanism are sufficient to guarantee asymptotic global convergence. The first condition is that the neighborhood operator must be able to move from any state to a globally optimal state in a finite number of transitions. The presence of mutation satisfies this requirement. The second condition is symmetry. It requires that the probability at any temperature of generating state y from state x is the same as the probability of generating state x from state y. Symmetry holds for common crossover operators such as single-point, multipoint, and uniform crossover (Mahfoud and Goldberg 1995).

References

Aarts E and Korst J 1989 *Simulated Annealing and Boltzmann Machines: a Stochastic Approach to Combinatorial Optimization and Neural Computing* (Chichester: Wiley)

Azencott R (ed) 1992 *Simulated Annealing: Parallelization Techniques* (New York: Wiley)

de la Maza M and Tidor B 1993 An analysis of selection procedures with particular attention paid to proportional and Boltzmann selection *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 124–31

Goldberg D E 1990 A note on Boltzmann tournament selection for genetic algorithms and population-oriented simulated annealing *Complex Syst.* **4** 445–60

Ingber L and Rosen B 1992 Genetic algorithms and very fast simulated re-annealing: a comparison *Math. Comput. Modelling* **16** 87–100

Kirpatrick S, Gelatt C D Jr and Vecchi M P 1983 Optimization by simulated annealing *Science* **220** 671–80

Mahfoud S W 1993 Finite Markov chain models of an alternative selection strategy for the genetic algorithm *Complex Syst.* **7** 155–70

——1995 *Niching Methods for Genetic Algorithms* Doctoral Dissertation and IlliGAL Report 95001, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory; *Dissertation Abstracts Int.* **56**(9) p 49878 *(University Microfilms 9543663)*

Mahfoud S W and Goldberg D E 1992 A genetic algorithm for parallel simulated annealing *Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature (Brussels, 1992)* ed R Männer and B Manderick (Amsterdam: Elsevier) pp 301–10

——1995 Parallel recombinative simulated annealing: a genetic algorithm *Parallel Comput.* **21** 1–28

Romeo F and Sangiovanni-Vincentelli A 1991 A theoretical framework for simulated annealing *Algorithmica* **6** 302–45

C2.6 Other selection methods

David B Fogel

Abstract

Selection methods not covered in other sections are introduced, including the tournament selection typically used in evolutionary programming, soft brood selection, and other methods.

C2.6.1 Introduction

In addition to the methods of selection presented in other sections of this chapter, other procedures for selecting parents of successive generations are of interest. These include the tournament selection typically used in evolutionary programming (Fogel 1995, p 137), soft brood selection offered within research in genetic programming (Altenberg 1994a, b), disruptive selection (Kuo and Hwang 1993), Boltzmann selection (de la Maza and Tidor 1993), nonlinear ranking selection (Michalewicz 1996), competitive selection (Hillis 1992, Angeline and Pollack 1993, Sebald and Schlenzig 1994), and the use of lifespan (Bäck 1996).

C2.6.2 Tournament selection

The tournament selection typically performed in evolutionary programming allows for tuning the degree of stringency of the selection imposed. Rather than selecting on the basis of each solution's fitness or error in light of the objective function at hand, selection is made on the basis of the number of *wins* earned in a competition. Each solution is made to compete with some number, q, of randomly selected solutions from the population. In each pairing, if the first solution's score is at least as good as the randomly selected opponent, the first solution receives a win. Thus up to q wins can be earned. This competition is conducted for all solutions in the population and selection then chooses the best subset of a given size from the population based on the number of wins each solution has earned. For $q = 1$, the procedure yields essentially a random walk with very low selection pressure. For $q = \infty$, the procedure becomes selection based on objective function scores (with no probabilistic selection). For practical purposes, $q \geq 10$ is often considered relatively *hard* selection, and q in the range of three to five is considered *soft*. Soft selection allows for lower probabilities of becoming trapped at local optima for periods of time.

C2.6.3 Soft brood selection

Soft brood selection holds a *tournament* (see C2.3) between members of a *brood* of two parents. The winner of the tournament is considered to be the offspring contributed by the mating. Soft brood selection is intended to shield the recombination operator from the costs of producing deleterious offspring. It culls such offspring, essentially testing for their viability before being placed into competition with the remainder of the population. (For further details on the effects of soft brood selection on subexpressions in tree structures, see the article by Altenberg (1994a).)

C2.3

C2.6.4 Disruptive selection

Disruptive selection can be used to select against individuals with moderate values (in contrast to stabilizing selection which acts against extreme values, or directional selection which acts to increase or decrease values). Kuo and Hwang (1993) suggested a fitness function of the form

$$u(\boldsymbol{x}) = |f(\boldsymbol{x}) - f(t)|$$

where $f(\boldsymbol{x})$ is the objective value of the solution $\boldsymbol{x}$ and $f(t)$ is the mean of all solutions in the population at time t. Thus a solution's fitness increases with its distance from the mean of all current solutions. The idea is to distribute more search effort to both the extremely good and extremely bad solutions. The utility of this method is certainly very problem dependent.

C2.6.5 Boltzmann selection

Boltzmann selection (as offered by de la Maza and Tidor 1993) proceeds as

$$F_i(U(X)) = \exp(U_i(X)/T)$$

where X is a population of solutions, $U(X)$ is the problem dependent objective function, $F_i(\cdot)$ is the fitness function for the ith solution in X, $U_i(\cdot)$ is the objective function evaluated for the ith solution in X, and T is a variable temperature parameter. De la Maza and Tidor (1993) suggest that this method of assigning fitness proportional selection converges faster than traditional proportional selection. Bäck (1994), however, describes this as a 'misleading name for yet another scaling method for proportional selection...'.

C2.6.6 Nonlinear ranking selection

Nonlinear ranking selection (Michalewicz 1996, pp 60–1) is a variant of linear ranking selection. Recall that for linear ranking selection, the probability of a solution with a given rank being selected can be set as

$$P(\text{rank}) = q - (\text{rank} - 1)r$$

where q is a user-defined parameter. For each lower rank, the probability of being selected is reduced by a factor of r. The requirement that the sum of all the probabilities for each ranked solution must be equal to unity implies that

$$q = r(\text{popsize} - 1)/2 + 1/\text{popsize}$$

where popsize is the number of solutions in the population. This relationship can be made nonlinear by setting:

$$P(\text{rank}) = q(1 - q)^{\text{rank}-1}$$

where $q \in (0, 1)$ and does not depend on popsize; larger values of q imply stronger selective pressure. Bäck (1994) notes that this nonlinear ranking method fails to sum to unity and can be made practically identical to tournament selection under the choice of q.

C2.6.7 Competitive selection

Competitive selection is implemented such that the fitness of a solution is determined by its interactions with other members of the population, or other members of a jointly evolving but separate population. Hillis (1992) used this concept to evolve sorting networks in which a population of sorting networks competed against a population of various permutations; the networks were scored in light of how well they sorted the permutations and the permutations were scored in light of how well they could defeat the sorting networks. Angeline and Pollack (1993) used a similar idea to evolve programs to play tic-tac-toe. Sebald and Schlenzig (1994) used evolutionary programming on competing populations to generate suitable blood pressure controllers for simulated patients undergoing cardiac surgery (i.e. controllers were scored on how well they maintained the patient's blood pressure while patients were scored on how well they defeated the controllers). Fogel and Burgin (1969) describe experiments in which competing evolutionary programs played a prisoner's dilemma game using finite-state machines, but insufficient detail is provided to allow for replication of the results. Axelrod (1987), and others, offered an apparently similar procedure for evolving rule sets describing alternative behaviors in the iterated prisoner's dilemma.

C2.6.8 Variable lifespan

Finally, Bäck (1996) notes that the concept of a variable lifespan has been incorporated into the (μ, λ) selection of evolution strategies by Schwefel and Rudolph (1995) by allowing the parents to survive some number of generations. When this number is one generation, the method is the familiar comma strategy; at infinity, the method becomes a plus strategy.

References

Altenberg L 1994a Emergent phenomena in genetic programming, *Proc. 3rd Ann. Conf. on Evolutionary Programming (San Diego, CA, February 1994)* ed A V Sebald and L J Fogel (Singapore: World Scientific) pp 233–41

——1994b The evolution of evolvability in genetic programming *Advances in Genetic Programming* ed K Kinnear (Cambridge, MA: MIT Press) pp 47–74

Axelrod R 1987 The evolution of strategies in the iterated prisoner's dilemma *Genetic Algorithms and Simulated Annealing* ed L Davis (Los Altos, CA: Morgan Kaufmann) pp 32–41

Angeline P J and Pollack J B 1993 Competitive environments evolve better solutions for complex tasks *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 264–70

Bäck T 1994 Selective pressure in evolutionary algorithms: a characterization of selection mechanisms *Proc. 1st IEEE Conf. on Evolutionary Computation (Orlando, FL, 1993)* (Piscataway, NJ: IEEE) pp 57–62

——1996 *Evolutionary Algorithms in Theory and Practice* (New York: Oxford University Press)

de la Maza M and Tidor B 1993 An analysis of selection procedures with particular attention paid to proportional and Boltzmann selection *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 124–31

Fogel D B 1995 *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence* (New York: IEEE)

Fogel L J and Burgin G H 1969 *Competitive Goal-Seeking Through Evolutionary Programming* Final Report, Contract No AF 19(628)-5927, Air Force Cambridge Research Labratories.

Hillis W D 1992 Co-evolving parasites improves simulated evolution as an optimization procedure *Artificial Life II* ed C Langton, C Taylor, J Farmer and S Rasmussen (Reading, MA: Addison-Wesley) pp 313–24

Kuo T and Hwang S-Y 1993 A genetic algorithm with disruptive selection *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 65–9

Michalewicz Z 1996 *Genetic Algorithms + Data Structures = Evolution Programs* 3rd edn (Berlin: Springer)

Schwefel H-P and Rudolph G 1995 Contemporary evolution strategies *Advances in Artificial Life (Proc. 3rd Int. Conf. on Artificial Life, Granada, Spain)* (*Lecture Notes in Artificial Intelligence 929*) ed F Morán *et al* (Berlin: Springer) pp 893–907

Sebald A V and Schlenzig J 1994 Minimax design of neural net controllers for highly uncertain plants *IEEE Trans. Neural Networks* **NN-5** 73–82

C2.7 Generation gap methods

Jayshree Sarma and Kenneth De Jong

Abstract

In this section, we will examine how generation gap methods are related to the concept of overlapping and nonoverlapping populations in evolutionary algorithms (EAs). We will then review why generation gap methods were designed. We will compare and contrast the generational and steady state systems. Finally we will describe the elitist strategies used in evolutionary algorithms.

C2.7.1 Introduction

The concept of a generation gap is linked to the notion of *nonoverlapping* and *overlapping* populations. In a nonoverlapping model parents and offspring never compete with one another, i.e. the entire parent population is always replaced by the offspring population, while in an overlapping system parents and offspring compete for survival. The term *generation gap* refers to the amount of overlap between parents and offspring. The notion of a generation gap is closely related to selection algorithms and population management issues.

A selection algorithm in an evolutionary algorithm (EA) involves two elements: (i) a selection pool and (ii) a selection distribution over that pool. A selection pool is required for reproduction selection as well as for deletion selection. The key issue in both these cases is 'what does the pool contain when parents are selected and when survivors are selected?'.

In the selection for the reproduction phase, parents are selected to produce offspring and the selection pool consists of the current population. How the parents are selected for reproduction depends on the individual EA paradigm.

In the selection for the deletion phase, a decision has to be made as to which individuals to select for deletion to make room for the new offspring. In nonoverlapping systems the entire selection pool consisting of the current population is selected for deletion: the parent population (μ) is always replaced by the offspring population (λ). In overlapping models, the selection pool for deletion consists of both parents and their offspring. Selection for deletion is performed on this combined set and the actual selection procedure varies in each of the EA paradigms.

Historically, both evolutionary programming and evolution strategies had overlapping populations while the canonical genetic algorithms used nonoverlapping populations.

C2.7.2 Historical perspective

In *evolutionary programming* (Fogel *et al* 1966), each individual produces one offspring and the best half from the parent and offspring populations are selected to form the new population. This is an overlapping system as the parents and their offspring constantly compete with each other for survival. B1.4

In *evolution strategies* (Schwefel 1981), the $(\mu + \lambda)$ and the (μ, λ) models correspond to the overlapping and nonoverlapping populations respectively. In the $(\mu + \lambda)$ system parents and offspring compete for survival and the best μ are selected. In the (μ, λ) model the number of offspring produced is generally far greater than the parents. The offspring are then ranked according to fitness and the best μ are selected to replace the parent population. B1.3

Genetic algorithms are based on the two reproductive plans introduced and analyzed by Holland (1975). In the first plan, R_1, at each time step a single individual was selected probabilistically using payoff proportional selection to produce a single offspring. To make room for this new offspring, one individual from the current population was selected for deletion using a uniform random distribution. B1.2

In the second plan, R_d, at each time step all individuals were deterministically selected to produce their expected number of offspring. The selected parents were kept in a temporary storage location. When the process of recombination was completed, the offspring produced replaced the entire current population. Thus in R_d, individuals were guaranteed to produce their expected number of offspring (within probabilistic roundoff).

At that time, from a theoretical point of view, the two plans were viewed as generally equivalent. However, because of practical considerations relating to the overhead of recalculating selection probabilities and severe genetic drift (allele loss) in small populations, most early researchers favored the R_d approach.

The earliest attempt at evaluating the properties of R_1 and R_d plans was a set of empirical studies (De Jong 1975) in which a parameter G, called the *generation gap*, was defined to introduce the notion of overlapping generations. The generation gap parameter controls the fraction of the population to be replaced in each generation. Thus, $G = 1$ (replacing the entire population) corresponded to R_d and $G = 1/\mu$ (replacing a single individual) represented R_1.

These early studies (De Jong 1975) suggested that any advantages that overlapping populations might have were offset by the negative effects of genetic drift (allele loss). The genetic drift was caused by the high variance in expected lifetimes and expected number of offspring, mainly because at that time, generally, modest population sizes were used ($\mu \leq 100$). These negative effects were shown to increase in severity as G was reduced. These studies also suggested the advantages of an *implicit* generation overlap. That is, using the optimal crossover rate of 0.6 and optimal mutation rate of 0.001 (identified empirically for the test suite used) meant that approximately 40% of the offspring were clones of their parents, even for $G = 1$. A later empirical study by Grefenstette (1986) confirmed the earlier results that a larger generation gap value improved performance.

However, early experience with *classifier systems* (see e.g. Holland and Reitman 1978) yielded quite the opposite behavior. In classifier systems only a subset of the population is replaced each time step. Replacing a small number of classifiers was generally more beneficial than replacing a large number or possibly all of them. Here the poor performance observed as the generation gap value increased was attributed to the fact that the population as a whole represented a single solution and thus could not tolerate large changes in its content. B1.5.2

In recent years, computing equipment with increased capacity is easily available and this effectively removes the reason for preferring the R_d approach. The desire to solve more complex problems using genetic algorithms has prompted researchers to develop an alternative to the generational system called the 'steady state' approach, in which typically parents and offspring do coexist (see e.g. Syswerda 1989, Whitley and Kauth 1988).

C2.7.3 Steady state and generational evolutionary algorithms

Steady state EAs are systems in which usually only one or two offspring are produced in each generation. The selection pool for deletion can consist of the parent population only or can be possibly augmented by the offspring produced. The appropriate number of individuals are selected for deletion, based on some distribution, to make room for these new offspring. Generational systems are so named because the entire population is replaced every generation by the offspring population: the lifetime of each individual in the population is only one generation. This is the same as the *nonoverlapping* population systems, while the steady state EA is an *overlapping* population system.

One can conceptually think of a steady state model in evolutionary programming and evolution strategies. For example, from a parent population of μ individuals, a single offspring can be formed by recombination and mutation and can then be inserted into the population. A recent study of the steady state evolutionary programming performed by Fogel and Fogel (1995) concluded that the generational model of evolutionary programming may be more appropriate for practical optimization problems. The first example of the steady state evolutionary strategies is the $(\mu + 1)$ approach introduced by Rechenberg (1973) which had a parent population greater than one ($\mu > 1$). All the parents were then allowed to participate in the reproduction phase to create one offspring. The $(\mu + 1)$ model was not used as it was not feasible to selfadapt the step sizes (Bäck *et al* 1991).

An early example of the steady state model of genetic algorithms is the R_1 model defined by Holland (1975) in which the selection pool for deletion consists only of the parent population and a uniform deletion strategy is used. The R_d approach is the generational genetic algorithm. Theoretically, the two systems (overlapping systems using uniform deletion and nonoverlapping systems) are considered to be similar in expectation for infinite populations. However, there can be high variance in the expected lifetimes and expected number of offspring when small finite populations are used.

This variance can be highlighted by keeping everything in the two systems constant and changing only one parameter, viz., the number of offspring produced. Figures C2.7.1 and C2.7.2 illustrate the average and variance for the growth curve of the best in two systems, producing and replacing only a single individual each generation in one and replacing the entire population each generation in the other. A population size of 50 was used, the best occupied 10% of the initial population, and the curves are averaged over 100 independent runs. Only payoff proportional selection, reproduction, and uniform deletion were used to drive the systems to a state of equilibrium. Notice that in the overlapping system (figure C2.7.1) the best individuals take over the population only about 80% of the time and the growth curves exhibit much higher variance when compared to the nonoverlapping population (figure C2.7.2).

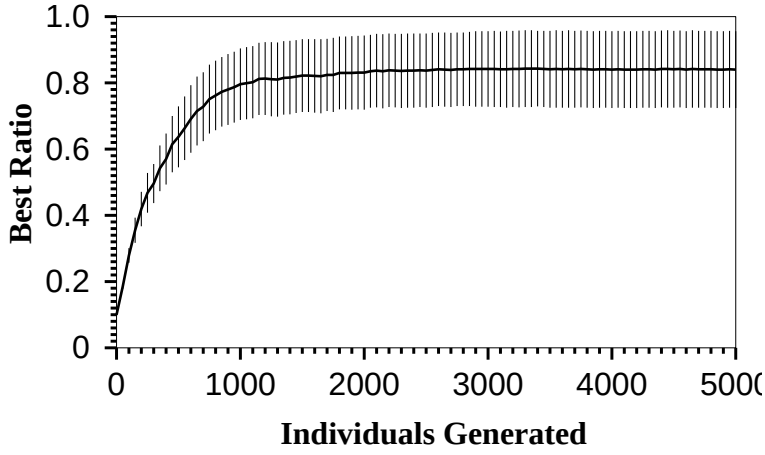


Figure C2.7.1. The mean and variance of the growth curves of the best in an overlapping system (population size, 50; $G = 1/50$).

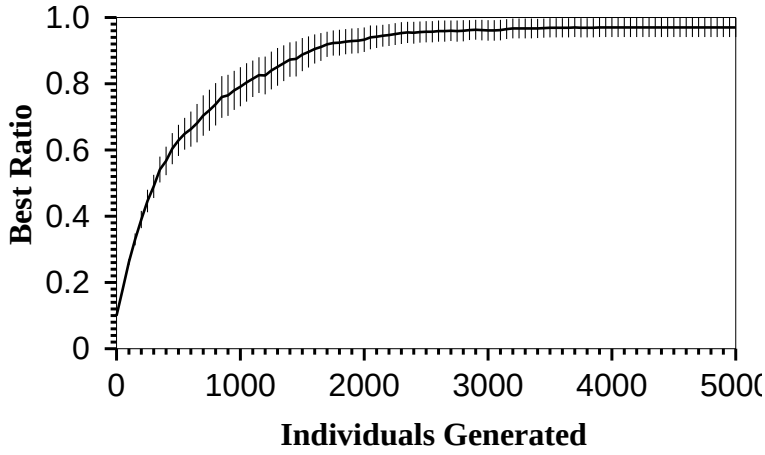


Figure C2.7.2. The mean and variance of the growth curves of the best in a nonoverlapping system (population size, 50; $G = 1$).

This high variance for small generation gap values causes more genetic drift (allele loss). Hence, with smaller population sizes, the higher variance in a steady state system makes it easier for alleles to disappear. Increasing the population size is one way to reduce the the variance (see figure C2.7.3) and thus

offset the allele loss. In summary, the main difference between the generational and steady state systems is higher genetic drift in the latter especially when small population sizes are used with low generation gap values. (See the article by De Jong and Sarma (1993) for more details.)

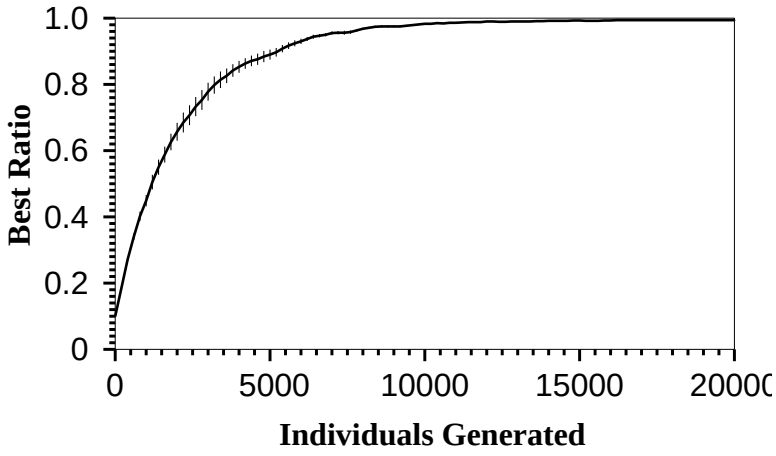


Figure C2.7.3. The mean and variance of the growth curves of the best in an overlapping system (population size, 200; $G = 1/200$).

So far we have assumed that there is an uniform distribution on the selection pool used for deletion, but most researchers using a steady state genetic algorithm generally use a distribution other than the standard uniform distribution. Syswerda (1991) shows how the growth curves can change when different deletion strategies, such as deleting the least fit, exponential ranking of the members in the selection pool, and reverse fitness, are used. Peck and Dhawan (1995) demonstrate an improvement in the ideal growth behavior of the steady state system when uniform deletion is changed to a first-in–first-out (FIFO) deletion strategy. An early model of a steady state (overlapping) system is GENITOR (Whitley and Kauth 1988, Whitley 1989) which not only uses *ranking selection* instead of *proportional selection* on the selection pool for reproduction but also uses deletion of the worst member as the deletion strategy. The GENITOR approach exhibited significant performance improvement over the standard generational approach. C2.2, C2.4

Using a deletion scheme other than a uniform deletion changes the selection pressure. The selection pressure induced by the different selection schemes can vary considerably. Both these changes can alter the exploration–exploitation balance. Two different studies have shown that improved performance in a steady state system, like GENITOR, is due to higher growth rates and changes in the exploration–exploitation balance caused by using different selection and deletion strategies and is not due to the use of an overlapping model (Goldberg and Deb 1991, De Jong and Sarma 1993).

C2.7.4 Elitist strategies

The cycle of birth and death of individuals is very much linked to the management of the population. Individuals that are born have an associated lifetime. The expected lifetime of an individual is typically one generation, but in some EA systems it can be longer. We now explore this issue in more detail.

Elitist strategies link the lifetimes of individuals to their fitnesses. Elitist strategies are techniques to keep good solutions in the population longer than one generation. Though all individuals in a population can expect to have a lifetime of one generation, individuals with higher fitness can have a longer lifetime when elitist strategies are used.

As stated earlier, the selection pool for deletion is comprised of both the parents and the offspring populations in the overlapping system. This combined population is usually ranked according to fitness and then truncated to form the new population. This method ensures that most of the current individuals with higher fitness survive into the next generation, thus extending their lifetime. In the $(\mu + \lambda)$ evolution strategies, a very strong elitist policy is in effect as the top μ are always kept. In evolutionary programming, a stochastic tournament is used to select the survivors, and hence the elitist policy is not quite as strong as in the evolution strategy case. In the (μ, λ) evolution strategies there is no elitist strategy to preserve the best parents.

Unlike evolution strategies and evolutionary programming, where there is postselection of survivors based on fitness, in generational genetic algorithms there is only preselection of parents for reproduction. Recombination operators are applied to these parents to produce new offspring, which are then subject to mutation. Since all parents are replaced each generation by their offspring, there is no guarantee that the individuals with higher fitness will survive into the next generation. An elitist strategy in generational genetic algorithms is a way of ensuring that the lifetime of the very best individual is extended beyond one generation. Thus, unlike evolutionary programming and evolution strategies, where more than just the best individual survive, in generational genetic algorithms generally only the best individual survives. Steady state genetic algorithms which use deletion schemes other than uniform random deletion have an implicit elitist policy and so automatically extend the lifetime of the higher-fitness individuals in the population.

It should be noted that the elitist strategies were deemed necessary when genetic algorithms are used as function optimizers and the goal is to find a global optimal solution (De Jong 1993). Elitist strategies tend to make the search more exploitative rather than explorative and may not work for problems in which one is required to find multiple optimal solutions.

References

Bäck T, Hoffmeister F and Schwefel H-P 1991 A survey of evolutionary strategies *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, July 1991)* ed R K Belew and L B Booker (San Mateo, CA: Morgan Kaufmann) pp 2–9

De Jong K A 1975 *An Analysis of the Behavior of a Class of Genetic Adaptive Systems* Phd Dissertation, University of Michigan

——1993 Genetic algorithms are NOT function optimizers *Foundations of Genetic Algorithms 2* ed L D Whitley (San Mateo, CA: Morgan Kaufmann) pp 5–17

De Jong K A and Sarma J 1993 Generation gaps revisited *Foundations of Genetic Algorithms 2* ed L D Whitley (San Mateo, CA: Morgan Kaufmann) pp 19–28

Fogel G B and Fogel D B 1995 Continuous evolutionary programming: analysis and experiments *Cybernet. Syst.* **26** 79–90

Fogel L J, Owens A J and Walsh M J 1996 *Artificial Intelligence through Simulated Evolution* (New York: Wiley)

Goldberg D E and Deb K 1991 A comparative analysis of selection schemes used in genetic algorithms *Foundations of Genetic Algorithms 1* ed G J E Rawlins (San Mateo, CA: Morgan Kaufmann) pp 69–93

Grefenstette J J 1986 Optimization of control parameters for genetic algorithms *IEEE Trans. Syst. Man Cybernet.* **SMC-16** 122–8

Holland J H 1975 *Adaptation in Natural and Artificial Systems* (Ann Arbor, MI: University of Michigan Press)

Holland J H and Reitman J S 1978 Cognitive systems based on adaptive algorithms *Pattern-Directed Inference Systems* ed D A Waterman and F Hayes-Roth (New York: Academic)

Peck C C and Dhawan A P 1995 Genetic algorithms as global random search methods: an alternative perspective *Evolutionary Comput.* **3** 39–80

Rechenberg I 1973 *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution* (Stuttgart: Frommann-Holzboog)

Schwefel H-P 1981 *Numerical Optimization of Computer Models* (Chichester: Wiley)

Syswerda G 1989 Uniform crossover in genetic algorithms *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, June 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 2–9

——1991 A study of reproduction in generational and steady-state genetic algorithms *Foundations of Genetic Algorithms 1* ed G J E Rawlins (San Mateo, CA: Morgan Kaufmann) pp 94–101

Whitley D 1989 The GENITOR algorithm and selection pressure: why rank-based allocation of reproductive trials is best *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, June 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 116–21

Whitley D and Kauth J 1988 *GENITOR: a Different Genetic Algorithm* Colorado State University Technical Report CS-88-101

C2.8 A comparison of selection mechanisms

Peter J B Hancock

Abstract

Selection methods differ in the way they distribute reproductive opportunities and the consistency with which they do so, and in various other ways, including their sensitivity to evaluation noise. These differences are demonstrated by reference to analytical and simulation studies of simplified systems, often including only selection, or selection and one genetic operator. A number of equivalences between apparently different algorithms are identified. The sensitivity of a number of common selection methods to their parameters is illustrated.

C2.8.1 Introduction

Selection provides the driving force behind an evolutionary algorithm. Without it, the search would be no better than random. This section explores the pros and cons of a variety of different methods of performing selection. Selection methods differ in two main ways: the way they aim to distribute reproductive opportunities across members of the population, and the accuracy with which they achieve their aim. The accuracy may differ because of sampling noise inherent in some selection algorithms. There are also other differences that may be significant, such as time complexity and suitability for parallel processing. Crucially for some applications, they also differ in their ability to deal with evaluation noise.

There have been a number of comparisons of different selection methods by a mixture of analysis and simulation, usually on deliberately simplified tasks. Goldberg and Deb (1991) considered a system with just two fitness levels, and studied the time taken for the fitter individuals to take over the population under the action of selection only, verifying their analysis with simulations. Hancock (1994) extended the simulations to a wider range of selection algorithms, and added mutation as a source of variation, to compare effective growth rates. The effects of adding noise to the evaluation function were also considered. Syswerda (1991) compared generational and incremental models on a ten-level takeover problem. Thierens and Goldberg (1994) derived analytical results for rates of growth for a bit counting problem, where the approximately normal distribution of fitness values allowed them to include recombination in their analysis. Bäck (1994) compared takeover times for all the major selection methods analytically and reported an experiment on a 30-dimensional sphere problem. Bäck (1995) compared tournament and (μ, λ) selection more closely. Blickle and Thiele (1995a, b) undertook a detailed analytical comparison of a number of selection methods (note that the second paper corrects an error in the first). Other studies include those of Bäck and Hoffmeister (1991), de la Maza and Tidor (1993) and Pál (1994).

It would be useful to have some objective measure(s) with which to compare selection methods. A general term is selection pressure. The meaning of this is intuitively clear, the higher the selection pressure, the faster the rate of convergence, but it has no strict definition. Analysis of selection methods has concentrated on two measures: takeover time and selection intensity. Takeover time is the number of generations required for one copy of the best string to reproduce so as to fill the population, under the effect only of selection (Goldberg and Deb 1991). Selection intensity is defined in terms of the average fitness before and after selection, $\bar{f}$ and $\bar{f}_{\text{sel}}$, and the fitness variance σ:

$$I = \frac{\bar{f}_{\text{sel}} - \bar{f}}{\sigma}.$$

This captures the notion that it is harder to produce a given step in average fitness between the population and those selected when the fitness variance is low. However, both takeover time and selection intensity depend on the fitness functions, and so theoretical results may not always transfer to a real problem. There is an additional difficulty because the fitness variance itself depends on the selection method, so different methods configured to have the same selection intensity may actually grow at different rates.

Most of the selection schemes have a parameter that controls either the proportion of the population that reproduces or the distribution of reproductive opportunities, or both. One aim in what follows will be to identify some equivalent parameter settings for different selection methods.

C2.8.2 Simulations

A number of graphs from simulations similar to those reported by Hancock (1994) are shown here, along with some analytical and experimental results from elsewhere. The takeover simulation initializes a population of 100 randomly, with rectangular distribution, in the range 0–1, with the exception that one individual is set to 1. The rate of takeover of individuals with the value 1 under the action of selection alone is plotted. Results reported are averaged over 100 different runs. The simulation is thus similar to that used by Goldberg and Deb (1991), but the greater range of fitness values allows investigation of the diversity maintained by the different selection methods. Since some of them produce exponential takeover in such conditions, a second set of simulations makes the problem slightly more realistic by adding mutation as a source of variation to be exploited by the selection procedures. This growth simulation initializes the population in the range 0–0.1. During reproduction, mutation with a Gaussian distribution, mean 0, standard deviation 0.02, is added to produce the offspring, subject to remaining in the range 0–1. Some plots show the value of the best member of the population after various numbers of evaluations, again averaged over 100 different runs. Other plots show the growth of the worst value in the population, which gives an indication of the diversity maintained in the population. Some selection methods are better at preserving such diversity: other things being equal, this seems likely to improve the quality of the overall search (Mühlenbein and Schlierkamp-Voosen 1995, Blickle and Thiele 1995b).

It should be emphasized that fast convergence on these tasks is not necessarily good: they are deliberately simple in an effort to illustrate some of the differences between selection methods and the reasons underlying them. Good selection methods need to balance exploration and exploitation. Before reporting results, we shall consider a number of more theoretical points of similarities and differences.

C2.8.3 Population models

There are two different points in the population cycle at which selection may be implemented. One approach, typical of *genetic algorithms* (GAs), is to choose individuals from the population for reproduction, usually in some way proportional to their fitness. These are then acted on by the chosen genetic operators to produce the next generation. The other approach, more typical of *evolution strategies* (ESs) and *evolutionary programming* (EP), is to allow all the members of the population to reproduce, and then select the better members of the extended population to go through to the next generation. This difference, of allowing all members to reproduce, is sometimes flagged as one of the key differences in approach between ES/EP and GAs. In fact the two approaches may be seen as equivalent once running, differing only in what is called the population. If the extended population typical of the ES and EP approach is labeled simply the population, then it may be seen that, as with the first approach, the best individuals are selected for reproduction and used to generate the new (extended) population. Looked at this way, it is the traditional GA approach that allows all members of the population at least some chance of reproduction, where the methods that use truncation selection restrict the number that are allowed to breed. There remains, however, a difference in philosophy: the traditional GA approach is reproduction according to fitness, while the truncation selection typical of the ES, EP, and breeder GA is more like survival of the fittest. There will also be a difference at startup, with ES/EP initializing μ individuals, while an equivalent GA initializes $\mu + \lambda$.

B1.2 B1.3 B1.4

C2.8.4 Equivalence: expectations and reality

A number of pairs of the common selection algorithms turn out to be, in some respects, equivalent. The equivalence, usually in expected outcome, can hide differences due to sampling errors, or behavior in the

presence of noise, that may cause significant differences in practice. This section considers some of these similarities and differences, in order to reduce the number that need be considered in detail in section C2.8.5.

C2.8.4.1 Tournament selection and ranking

Goldberg and Deb (1991) showed that simple binary *tournament selection* (TS) is equivalent to *linear ranking* when set to give two offspring to the top-ranked string ($\beta_{\text{rank}} = 2$). However, this is only in expectation: when implemented the obvious way, picking each fresh pair of potential parents from the population with replacement, tournament selection suffers from sampling errors like those produced by roulette wheel sampling, precisely because each tournament is performed separately. A way to reduce this noise is to take a copy of the population and choose pairs for tournament from it *without* replacement. When the copy population is exhausted, another copy is made to select the second half of the new population (Goldberg *et al* 1989). This method ensures that each individual participates in exactly two tournaments, and will not fight itself. It does not eliminate the problem, since, for example, an average individual, that ought to win once, may pick better or worse opponents both times, but it will at least stop several copies of any one being chosen. C2.3, C2.4.2

The selection pressure generated by tournament selection may be decreased by making the tournaments stochastic. The equivalence, apart from sampling errors, with linear ranking remains. Thus TS with a probability of the better string winning of 0.75 is equivalent to linear ranking with $\beta_{\text{rank}} = 1.5$. The selection pressure may be increased by holding tournaments among more than two individuals. For three, the best will expect three offspring, while an average member can expect 0.75 (it should win one quarter of its expected three tournaments). The assignment is therefore nonlinear and Bäck (1994) shows that, to a first approximation, the results are equivalent to exponential nonlinear ranking, where the probability of selection of each rank i, starting at $i = 1$ for the best, is given by $(s-1)(s^{i-1})/(s^{\mu}-1)$, where s is typically in the range 0.9–1 (Blickle and Thiele 1995b). (Note that the probabilities as specified by Michalewicz (1992) do not sum to unity (Bäck 1994).) More precisely, they differ in that TS gives the worst members of the population no chance to reproduce. Figure C2.8.1 compares the expected number of offspring for each rank in a population of 100. The difference results in a somewhat lower population diversity for TS when run at the same growth rate.

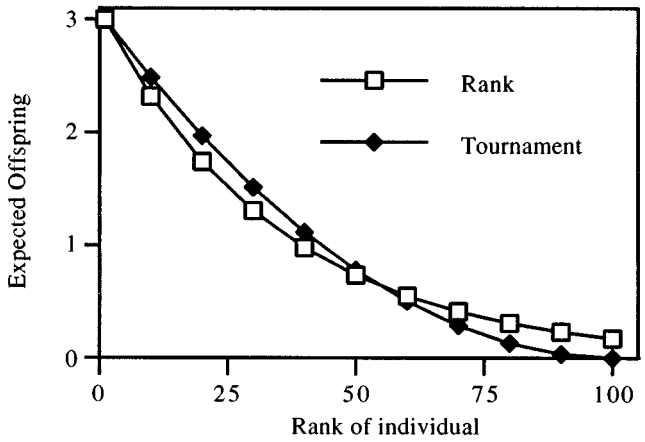


Figure C2.8.1. Expected number of offspring against rank for tournament selection with tournament size 3 and exponential rank selection with $s = 0.972$.

Goldberg and Deb (1991) prefer TS to linear ranking on account of its lower time complexity (since ranking requires a sort of the population), and Bäck (1994) argues similarly for TS over nonlinear ranking. However, time complexity is unlikely to be an issue in serious applications, where the evaluation time usually dominates all other parts of the algorithm. The difference is in any case reduced if the noise-reduced version of TS is implemented, since this also requires shuffling the population. For global population models, therefore, ranking, with Baker's sampling procedure (Baker 1987), is usually preferable. TS may be appropriate in incremental models, where only one individual is to be evaluated at a time, and in

parallel population models. It may also be appropriate in, for instance, game playing applications, where the evaluation itself consists of individuals playing each other.

Freisleben and Härtfelder (1993) compared a number of selection schemes using a meta-level GA, that adjusted the parameters of the GA used to tackle their problem. Tournament selection was chosen in preference to rank selection, which at first sight seems odd, since the only difference is added noise. A possible explanation lies in the nature of their task, which was learning the weights for a neural net simulation. This is plagued with symmetry problems (e.g. Hancock 1992). The GA has to break the symmetries and decide on just one to make progress. It seems possible that the inaccuracies inherent in tournament selection facilitated this symmetry breaking, with one individual having an undue advantage, and thereby taking over the population. Noise is not always undesirable, though there may be more controlled ways to achieve the same result.

C2.8.4.2 Incremental and generational models

There is apparently a large division between incremental and generational reproduction models. However, Syswerda (1991) shows that an incremental model where the deletion is at random produces the same expected result as a generational model with the same rank selection for reproduction. Again, however, this analysis overlooks sampling effects. Precisely because incremental models generate only one or two offspring per cycle, they suffer the roulette wheel sampling error. Figure C2.8.2 shows the growth rate for best and worst in the population for the two models with the same selection pressure (best expecting 1.2 offspring). The incremental model grows more slowly, yet loses diversity more rapidly, an effect characteristic of this kind of sampling error. Incremental models also suffer in the presence of evaluation noise (see section C2.8.6).

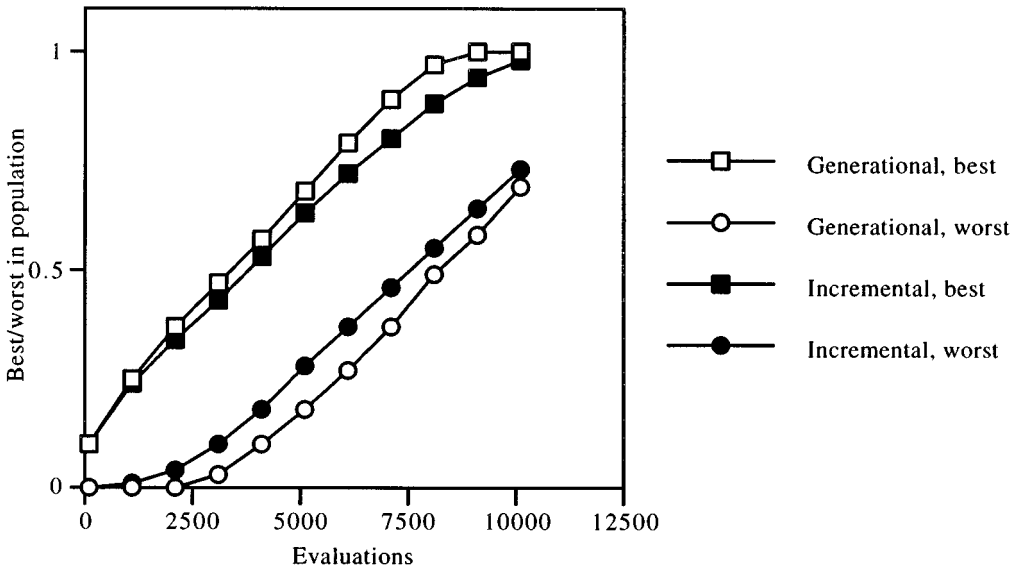


Figure C2.8.2. The growth rate in the presence of mutation of the best and worst in the population for the incremental model with random deletion and the generational model, both with linear rank selection for reproduction, $\beta_{\text{rank}} = 1.2$.

The very highest selection pressure possible from an evolutionary system would arise from an incremental system, where only the best member of the population is able to reproduce, and the worst is removed if the new string is an improvement. Since the rest of the population would thus be redundant, this is equivalent to a $(1 + 1)$ ES, the dynamics of which are well investigated (Schwefel 1981).

C2.8.4.3 Evolution strategy, evolutionary programming, and truncation selection

Some GA workers allow only the top few members of the population to reproduce (Nolfi *et al* 1990, Mühlenbein and Schlierkamp-Voosen 1993). This is often called truncation selection, and is equivalent to the ES (μ, λ) approach subject only to a difference in what is called the population (see section C2.8.3).

EP uses a form of tournament selection where all members of the extended population $\mu+\lambda$ compete with c others, chosen at random with replacement. Those μ that amass the most wins then reproduce by mutation to form the next extended population. This may be seen as a rather softer form of truncation selection, converging to the same result as a $(\mu+\mu)$ ES as the size of c increases. The value of c does not directly affect the selection pressure, only the noise in the selection process.

The EP selection process may be softened further by making the tournaments probabilistic. One approach is to make the probability of the better individual winning dependent on the relative fitness of the pair: $p_i = f_i/(f_i + f_j)$ (Fogel 1988). Although intuitively appealing, this has the effect of reducing selection pressure as the population converges and can produce growth curves remarkably similar to unscaled fitness proportional selection (FPS; Hancock 1994).

C2.8.5 Simulation results

C2.8.5.1 Fitness proportional selection

Simple FPS suffers from sensitivity to the distribution of fitness values in the population, as discussed in Section C2.2. The reduction of selection pressure as the population converges may be countered by moving baseline techniques, such as windowing and sigma scaling. These are still vulnerable to undesirable loss of diversity caused by a particularly fit individual, which may produce many offspring. Rescaling techniques are able to limit the number of offspring given to the best, but may still be affected by the overall spread of fitness values, and particularly by the presence of very poor individuals.

C2.2

Figure C2.8.3 compares takeover and growth rates of FPS and some of the baseline adjustment and rescaling methods. The simple takeover rates for the three adjusted methods are rather similar for these scale parameters, with linear scaling just fastest. Simple FPS is so slow it does not really show on the same graph: it reaches only 80% convergence after 40 000 evaluations on this problem. The curves for growth in the presence of mutation are all rather alike: the presence of the mutation maintains the range of fitness values in the population, giving simple FPS something to work on. Note, however, that it still starts off relatively fast and slows down towards the end: probably the opposite of what is desirable. The three scaled versions are still similar, but note that the order has reversed. Windowing and sigma scaling now grow more rapidly precisely because they fail to limit the number of offspring to especially good individuals. A fortuitous mutation is thus better exploited than in the more controlled linear scaling, which leads to the correct result in this simple hill-climbing task, but may not in a more complex real problem.

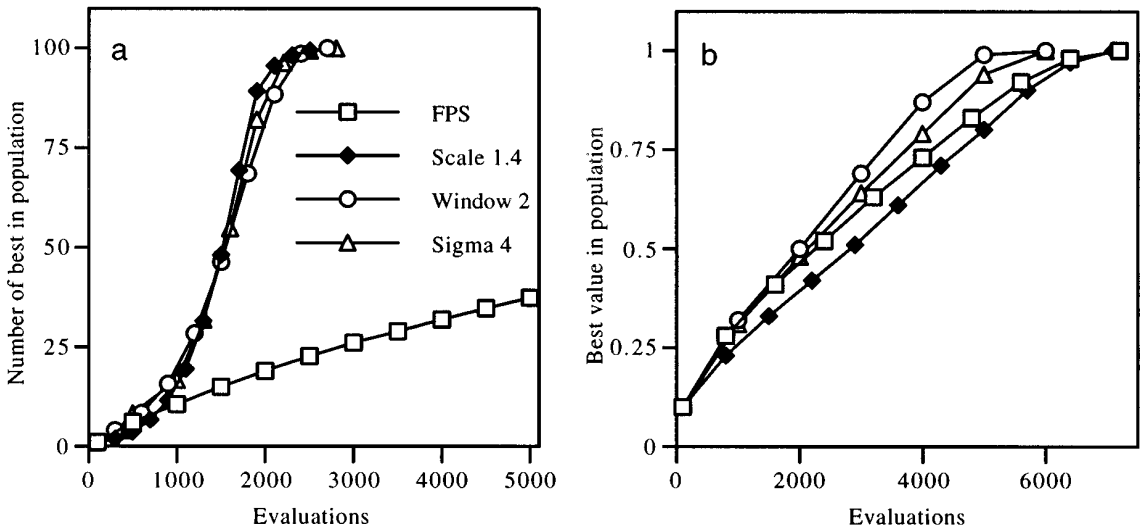


Figure C2.8.3. (a) The takeover rate for FPS, with windowing, sigma, and linear scaling. (b) Growth rates in the presence of mutation.

C2.8.5.2 Ranking

Goldberg and Deb (1991) show that the expected growth rate for linear ranking is proportional to the value of β_{rank}, the number of offspring given to the best individual. For exponential scaling, the selection pressure is proportional to $1 - s$. This makes available a wide range of selection pressures, defined by the value of s, illustrated in figure C2.8.4. The highest takeover rate available with linear ranking ($\beta_{\text{rank}} = 2$) is also shown. Exponential ranking can go faster with smaller values of s (see table C2.8.1). Note the logarithmic x-axis on this plot.

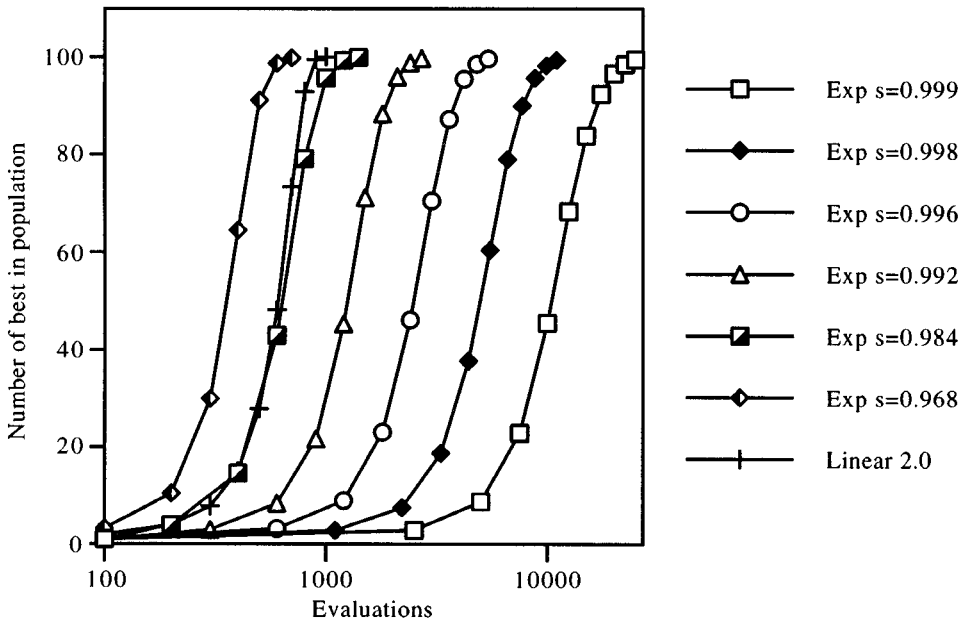


Figure C2.8.4. The takeover rate for exponential rank selection for a number of values of s, together with that for linear ranking, $\beta_{\text{rank}} = 2$.

With exponential ranking, because of the exponential assignment curve, poor individuals do rather better than with linear ranking, at the expense of those more in the middle of the range. One result of this is that, for parameter settings that give similar takeover times, exponential ranking loses the worse values in the population more slowly, which may help preserve diversity in practice.

C2.8.5.3 Evolution strategies

The selection pressure generated by the ES selection methods have been extensively analyzed, sometimes under the title of truncation selection (see e.g. Bäck 1994). Selection pressure is dependent on the ratio of μ to λ (see table C2.8.1). One simulation result is shown, in figure C2.8.5, to make clear the selection pressure achievable by (μ, λ) selection, and indicate its potential susceptibility to evaluation noise, discussed further below.

C2.8.5.4 Incremental models

Goldberg and Deb (1991) show that the Genitor incremental model develops a very high growth rate, compared to that typical of GAs. This is mostly due to the method of deletion, in which the worst member of the population is eliminated (cf the ES truncation approach). As a consequence, the value of β_{rank} used in the linear ranking to select parents has little effect on the takeover rate. Even with $\beta_{\text{rank}} = 1$ (i.e. a random choice of parents), kill-worst converges in around 900 evaluations (cf about 3000 for the scaled FPS variants in figure C2.8.3(a)). Increasing β_{rank} to its maximum of 2.0 only reduces this to around 600 evaluations.

There are a number of ways to decide which of the population should be removed (Syswerda 1991), such as killing the oldest (also known as FIFO deletion (De Jong and Sarma 1993)); one of the n worst;

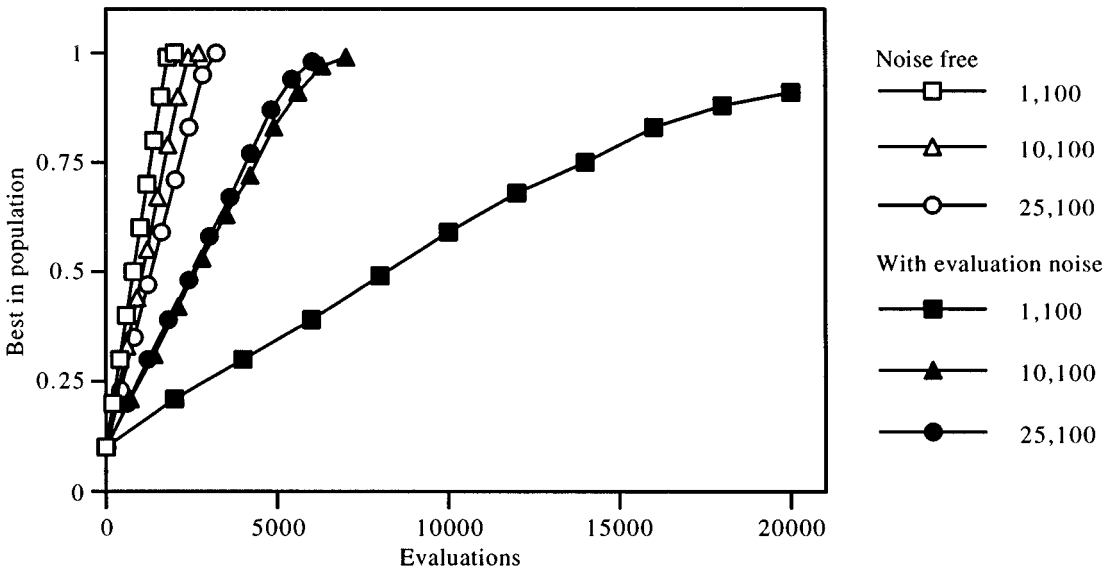


Figure C2.8.5. The growth rate in the presence of mutation for ES (μ, λ) selection with and without evaluation noise, for $\lambda = 100$ and $\mu = 1, 10,$ and 25.

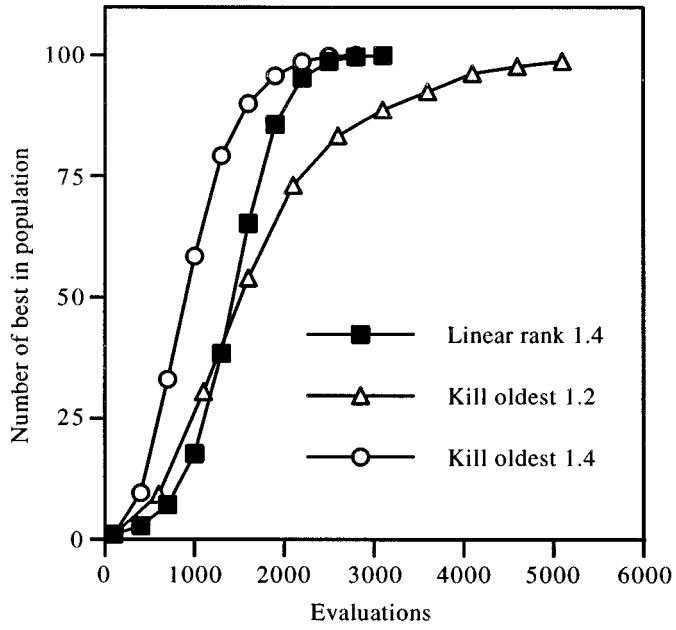


Figure C2.8.6. The takeover rates for the generational model and the kill-oldest incremental model, both using linear ranking for selection.

by inverse rank; or simply at random. The various deletion strategies radically affect the behavior of the algorithm. As discussed above, random deletion resembles a generational model. Kill oldest also produces much softer selection than kill-worst, producing takeover rates similar to generational models with the same selection pressure (see figure C2.8.6). However, the incremental model starts more quickly and ends more slowly than the generational one.

Syswerda (1991) prefers kill-by-inverse-rank. In his simulations, this produces results similar to kill-worst, but he is using a high inverse selection pressure (exponential ranking with $s = 0.9$). A more controlled result is given by selecting for reproduction from the top and for deletion from the bottom using ranking with the same, more moderate value of β_{rank}. Using linear ranking, the growth rate changes more rapidly than β_{rank}. This is because an increase in β_{rank} has two effects: increasing the probability of picking

one of the better members of the population at each step, and increasing the number of steps for which they are likely to remain in the population, by decreasing their probability of deletion. Figure C2.8.7 compares growth rates in the presence of mutation for kill-by-rank incremental and equivalent generational models. It may be seen that the generational model with $\beta_{\text{rank}} = 1.4$ and the incremental model with $\beta_{\text{rank}} = 1.2$ produce very similar results. Another matched pair at lower growth rates is generational with $\beta_{\text{rank}} = 1.2$ and incremental with $\beta_{\text{rank}} = 1.13$ (not shown).

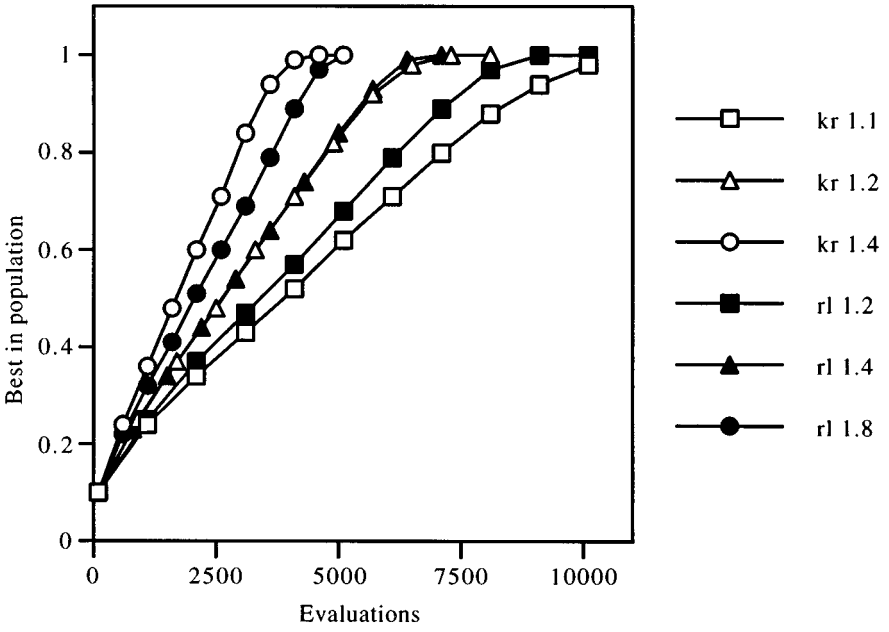


Figure C2.8.7. Growth rates in the presence of mutation for incremental kill-by-inverse-rank (kr) and generational linear ranking (rl) for various values of β_{rank}.

One of the arguments in favor of incremental models is that they allow good new individuals to be exploited at once, rather than having to wait a generation. It might be thought that any such gain would be rather slight, since although a good new member could be picked at once, it is more likely to have to wait several iterations at normal selection pressures. There is also the inevitable sampling noise to be overcome. De Jong and Sarma (1993) claim that there is actually no net benefit, since adding new fit members has the effect of increasing the average fitness, thus reducing the likelihood of them being selected. However, this argument applies only to takeover problems: when reproduction operators are included the incremental approach can generate higher growth rates. Figure C2.8.8 compares the growth of an incremental kill-oldest model with a generational model using the same selection scheme. The graph also shows one of the main drawbacks of the incremental models: their sensitivity to evaluation noise, to be discussed in the following section.

C2.8.6 The effects of evaluation noise

Hancock (1994) extended the growth simulations to study the effects of adding evaluation noise. A Gaussian random variable, mean zero, standard deviation 0.2, was added to each underlying true value for use in selection. The true value was used for reproduction. It proved necessary to add this much noise—ten times the standard deviation of the 'signal' mutation—to bring about significant reduction in growth rates for the generational selection models.

The sensitivity of the different selection algorithms to evaluation noise is largely dependent on whether they retain parents for further reproduction. Fully generational models are relatively immune, while most incremental models and those like the $(\mu + \lambda)$ ES that allow parents to compete for retention fare much worse, because individuals receiving a fortuitously good evaluation will be kept. The exception for incremental models is kill-oldest, which maintains the necessary turnover. Figure C2.8.8 shows the comparison. Kill oldest deteriorates only a little more than the generational model in the presence of noise, while kill-worst, which grows much the fastest in the absence of noise, almost fails completely.

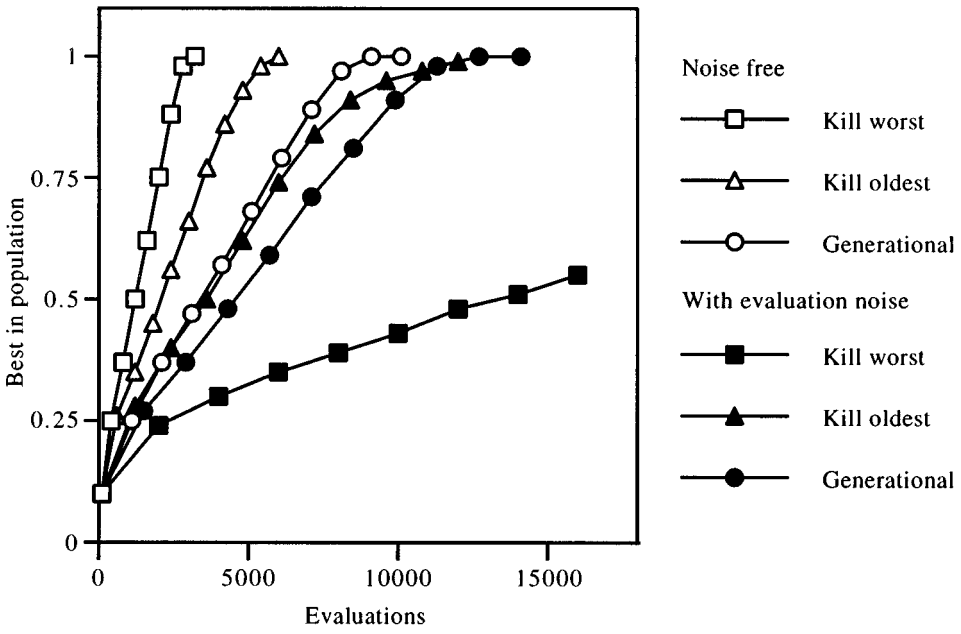


Figure C2.8.8. Growth in the presence of mutation, with and without evaluation noise, for the generational model with linear ranking and incremental models with kill-worst and kill-oldest, all using $\beta_{\text{rank}} = 1.2$ for selection.

Within generational models, there are differences in noise sensitivity. Figure C2.8.9 compares the growth rates for linear ranking and sigma-scaled FPS, with and without noise. It may be seen that the scaled FPS deteriorates less. This is caused by sigma scaling's inability to control superfit individuals. A genuinely good individual, that happens to receive a helpful boost from the noise, may be given many offspring by sigma scaling, but will be limited to β_{rank}, in this case 1.8, by ranking. As before, rapid convergence is beneficial in this simple task, but is unlikely to be so in general.

The ES (μ, λ) method can achieve extremely high selection pressures, when it becomes sensitive to evaluation noise in a manner similar to incremental models (figure C2.8.5). In this case, the problem is that too many reproductions are given to a few strings, whose rating may be overestimated by the noise. Figure C2.8.5 shows a clear turnaround: as the selection pressure is increased, performance in noise becomes worse.

One approach to countering the effects of noise is to perform two or more evaluations per string, and average the results. Fitzpatrick and Grefenstette (1988) investigated this and concluded that it is better to evaluate only once and proceed with the next generation. A possibly more efficient method is to reevaluate only the apparently fitter individuals. Candidates may be chosen as for reproduction, e.g. by rank. However, experiments with incremental kill-by-rank indicated that the extra evaluations did not pay their way, with convergence taking only a little less than twice as many evaluations in total (Hancock 1994). Hammel and Bäck (1994) compared the effects of reevaluation with an equivalent increase in the population size and showed that reevaluations lead to a better final result. Indeed, on Rastrigan's function, increasing the population size resulted in a deterioration of convergence performance. Taken together, these results suggest a strategy of evaluating only once initially, and keeping the population turning over, but then starting to reevaluate as the population begins to converge. Hammel and Bäck suggest an alternative possibility of incorporating reevaluation as a parameter to be optimized by the evolutionary system itself.

C2.8.7 Analytic comparison

Blickle and Thiele (1995b) perform an extensive analysis of several selection schemes, deriving the dependence of selection intensity on the selection parameters under the assumption of a normal distribution of fitness. Their results, which reassuringly agree with the simulation results here, are shown in an adapted form in table C2.8.1. They also consider selection variance, confirming that methods such as ES selection that disallow weakest strings from reproduction reduce the population variance more rapidly than those

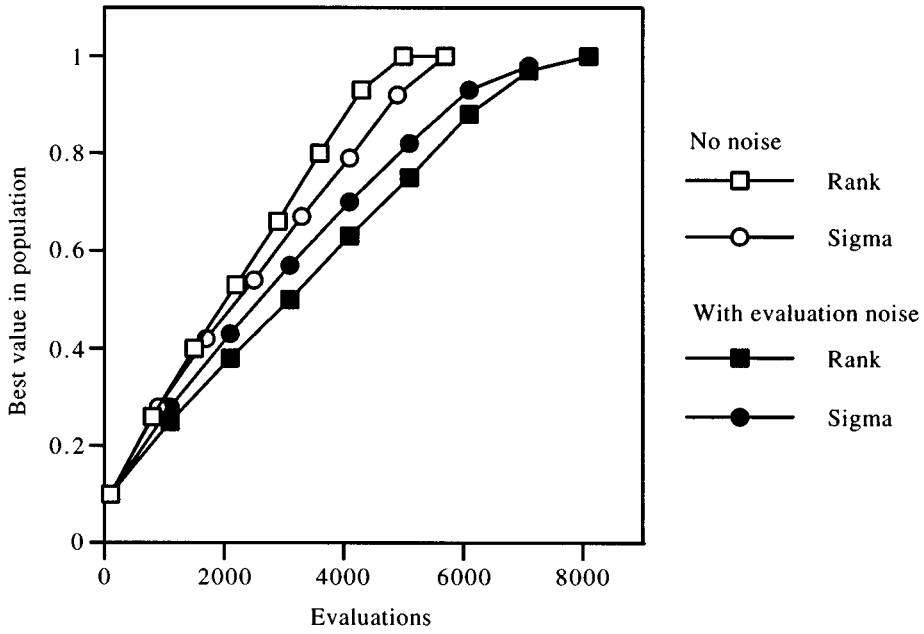


Figure C2.8.9. Growth in the presence of mutation, with and without evaluation noise, for the generational model with linear ranking, $\beta_{\text{rank}} = 1.8$, and sigma-scaled FPS, $s = 4$.

Table C2.8.1. Parameter settings that give equivalent selection intensities for ES (μ, λ), TS, and linear and exponential ranking, adapted and extended from Blickle and Thiele (1995b). Under tournament size, p refers to the probability of the better string winning.

I	ES μ/λ	Tournament size	β_{rank}, Lin rank	s, Exp rank, $\lambda = 100$
0.11	0.94	2, $p = 0.6$	1.2	0.996
0.34	0.80	2, $p = 0.8$	1.6	0.988
0.56	0.66	2	2.0	0.979
0.84	0.47	3	—	0.966
1.03	0.36	4	—	0.955
1.16	0.30	5	—	0.945
1.35	0.22	7	—	0.926
1.54	0.15	10	—	0.900
1.87	0.08	20	—	0.809

that allow weak strings some chance. Of the methods considered, exponential rank selection gives the highest fitness variance, for the reasons illustrated in figure C2.8.1. Their conclusion is that exponential rank selection is therefore probably the 'best' of the schemes that they consider.

C2.8.8 Conclusions

The choice of a selection mechanism cannot be made independently of other aspects of the evolutionary algorithm. For instance, Eshelman (1991) deliberately combines a conservative selection mechanism with an explorative recombination operator in his CHC algorithm. Where search is largely driven by mutation, it may be possible to use much higher selection pressures, typical of the ES approach. If the evaluation function is noisy, then most incremental models and others that may retain parents are likely to suffer. Certainly, selection pressures need to be lower in the presence of noise, and, of the incremental models, kill-oldest fares best. Without noise, incremental methods can provide a useful increase in exploitation of good new individuals. Care is needed in the choice of method of deletion: killing the worst provides high growth rates with little means of control. Killing by inverse rank or killing the oldest offers more control. Amongst generational models, the ES (μ, λ) and exponential rank selection methods give the biggest and most controllable range of selection pressures, with the ES method probably most suited to

mutation-driven, high-growth-rate systems, and ranking better for slower, more explorative searches, where maintenance of diversity is important.

References

Bäck T 1994 Selective pressure in evolutionary algorithms: a characterization of selection methods *Proc. 1st IEEE Conf. on Evolutionary Computation (Orlando, FL, June 1994)* (Piscataway, NJ: IEEE) pp 57–62

——1995 Generalized convergence models for tournament and (μ, λ)-selection *Proc. 6th Int. Conf. on Genetic Algorithms (Pittsburgh, PA, July 1995)* ed L J Eshelman (San Mateo, CA: Morgan Kaufmann) pp 2–8

Bäck T and Hoffmeister F 1991 Extended selection mechanisms in genetic algorithms *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, July 1991)* ed R K Belew and L B Booker (San Mateo, CA: Morgan Kaufmann) pp 92–9

Baker J E 1987 Reducing bias and inefficiency in the selection algorithm *Proc. 2nd Int. Conf. on Genetic Algorithms (Cambridge, MA, 1987)* ed J J Grefenstette (Hillsdale, NJ: Erlbaum) pp 14–21

Blickle T and Thiele L 1995a A mathematical analysis of tournament selection *Proc. 6th Int. Conf. on Genetic Algorithms (Pittsburgh, PA, July 1995)* ed L J Eshelman (San Mateo, CA: Morgan Kaufmann) pp 9–16

——1995b *A Comparison of Selection Schemes used in Genetic Algorithms* TIK-Report 11, Computer Engineering and Communication Networks Laboratory (TIK), Swiss Federal Institute of Technology

De Jong K A and Sarma J 1993 Generation gaps revisited *Foundations of Genetic Algorithms 2* ed D Whitley (San Mateo, CA: Morgan Kaufmann) pp 19–28

de la Maza M and Tidor B 1993 An analysis of selection procedures with particular attention paid to proportional and Boltzman selection *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 124–31

Eshelman L J 1991 The CHC adaptive search algorithm: how to have safe search when engaging in nontraditional genetic recombination *Foundations of Genetic Algorithms* ed G J E Rawlins (San Mateo, CA: Morgan Kaufmann) pp 265–83

Fitzpatrick J M and Grefenstette J J 1988 Genetic algorithms in noisy environments *Machine Learning* **3** 101–20

Fogel D B 1988 An evolutionary approach to the traveling salesman problem *Biol. Cybern.* **60** 139–44

Freisleben B and Härtfelder M 1993 Optimization of genetic algorithms by genetic algorithms *Artificial Neural Nets and Genetic Algorithms* ed R F Albrecht, C R Reeves and N C Steele (Berlin: Springer) pp 392–9

Goldberg D E and Deb K 1991 A comparative analysis of selection schemes used in genetic algorithms *Foundations of Genetic Algorithms* ed G J E Rawlins (San Mateo, CA: Morgan Kaufmann) pp 69–93

Goldberg D E, Korb B and Deb K 1989 Messy genetic algorithms: motivation, analysis and first results *Complex Syst.* **3** 493–530

Hammel U and Bäck T 1994 Evolution strategies on noisy functions: how to improve convergence properties *Parallel Problem solving from Nature—PPSN III (Proc. Int. Conf. on Evolutionary Computation and 3rd Conf. on Parallel Problem Solving from Nature, Jerusalem, October 1994)* (*Lecture Notes in Computer Science 866*) ed Yu Davidor, H-P Schwefel and R Männer (Berlin: Springer) pp 159–68

Hancock P J B 1992 *Coding strategies for genetic algorithms and neural nets* PhD Thesis, Department of Computer Science, University of Stirling

——1994 An empirical comparison of selection methods in evolutionary algorithms *Evolutionary Computing (AISB Workshop, Leeds, 1994, Selected Papers) (Lecture Notes in Computer Science 865)* ed T C Fogarty (Berlin: Springer) pp 80–94

Michalewicz Z 1992 *Genetic Algorithms + Data Structures = Evolution Programs* (Berlin: Springer)

Mühlenbein H and Schlierkamp-Voosen D 1993 Predictive models for the breeder genetic algorithm *Evolut. Comput.* **1** 25–50

——1995 Analysis of selection, mutation and recombination in genetic algorithms *Evolution as a Computational Process (Lecture Notes in Computer Science 899)* ed W Banzhaf and F H Eckman (Berlin: Springer) pp 188–214

Nolfi S, Elman J L and Parisi D 1990 *Learning and Evolution in Neural Networks* Technical report CRL TR 9019 UCSD

Pál K F 1994 Selection schemes with spatial isolation for genetic optimization *Parallel Problem solving from Nature—PPSN III (Proc. Int. Conf. on Evolutionary Computation and 3rd Conf. on Parallel Problem Solving from Nature, Jerusalem, October 1994)* (*Lecture Notes in Computer Science 866*) ed Yu Davidor, H-P Schwefel and R Männer (Berlin: Springer) pp 170–9

Schwefel H-P 1981 *Numerical Optimization of Computer Models* (Chichester: Wiley)

Syswerda G 1991 A study of reproduction in generational and steady-state genetic algorithms *Foundations of Genetic Algorithms* ed G J E Rawlins (San Mateo, CA: Morgan Kaufmann) pp 94–101

Thierens D and Goldberg D E 1994 Elitist recombination: an integrated selection recombination GA *Proc. IEEE World Congr. on Computational Intelligence* (Piscataway, NJ: IEEE)

C2.9 Interactive evolution

Wolfgang Banzhaf

Abstract

We present a different approach to directing the evolutionary process through interactive selection of solutions by the human user. First the general context of interactive evolution is set, then the standard interactive evolution algorithm is discussed together with more complicated variants. Finally, several application areas are discussed and uses for the new method are exemplified using design from the literature.

C2.9.1 Introduction

The basic idea of interactive evolution (IE) is to involve a human user on-line in the variation–selection loop of the evolutionary algorithm (EA). This is to be seen in contrast to the conventional participation of the user prior to running the EA by defining a suitable *representation* of the problem, the *fitness criterion* for evaluation of individual solutions, and corresponding *operators* to improve fitness quality. In the latter case, the user's role is restricted to passive observation during the EA run. C1, B2.1 C3

The minimum requirement for IE is the definition of a problem representation, together with a determination of *population parameters* only. *Search operators* of arbitrary kind as well as selection according to arbitrary criteria might be applied to the representation by the user. The process is much more comparable to the creation of a piece of art, for example, a painting, than to the automatic evolution of an optimized problem solution. In IE, the user assumes an active role in the search process. At the minimum level, the IE system must hold present solutions together with variants presently generated or considered. E1,C3

Usually, however, automatic means of variation (i.e. evolutionary search operators using random events) are provided with an IE system. In the present context we shall require the existence of automatic means of variation by operators for *mutation* and *recombination* of solutions which are to be defined prior to running the EA. C3.2, C3.3

C2.9.2 History and prospects

Dawkins (1986) was the first to consider an elaborate IE system. The evolution of *biomorphs*, as he called them, by IE in a system that he had originally intended to be useful for the design of treelike graphical forms has served as a prototype for many systems developed subsequently. Starting with the contributions of Sims (1991) and the book of Todd and Latham (1992), computer art developed into the present major application area of IE.

Later, IE of grammar-based structures has been considered (Nguyen and Huang 1994, McCormack 1994). Raw image data have been used more recently for the purpose of evolving forms (Graf and Banzhaf 1995a). It is anticipated that IE systems for the purpose of engineering design will be moving into application in the second half of the 1990s.

C2.9.3 The problem

The problem IE is trying to address has been encountered in all varieties of EAs that make use of automatic evolution: the existence of nonexplicit conditions, that is, conditions that are not formalizable.

- The absence of a human user in steering and controlling the process of evolution sometimes leads to unnecessary detours from the goal of global optimization. In most of these cases, human intervention into the search and selection process would advance the search rather quickly and allow faster *convergence* onto the most promising regions of the *fitness landscape*, or, sometimes, escape from a local optimum. Hence, a mobilization of human knowledge can be achieved by allowing the user to participate in the process. B2.3, B2.7
- Many design processes require subjective judgment relying on human intuition, aesthetical values, or taste. In such cases, the fitness criterion cannot be formulated explicitly, but can only be applied on a comparative case-by-case basis. Direct human participation in IE allows for machine-supported evolution of designs that would otherwise be completely manual.

Thus, IE can be used (i) to accelerate EAs and (ii) in some areas to enable application of EAs altogether.

C2.9.4 The interactive evolution approach

Selection in a standard IE system, as opposed to that in an automatic evolution system, is based on user action. It is typically the selection step that is subjugated to human action, although in less frequent cases the variation process might also be done by hand.

The standard algorithm for IE reads (following the notation in the introduction):

Input: $\mu, \lambda, \Theta_\iota, \Theta_m, \Theta_r, \Theta_s$
Output: a^*, the individual last selected during the run, or
P^*, the population last selected during the run.

1 $t \leftarrow 0$;
2 $P(t) \leftarrow$ initialize(μ);
3 **while** ($\iota(P(t), \Theta_\iota) \neq$ true) **do**
4 **Input:** Θ'_r, Θ'_m
5 $P'(t) \leftarrow$ recombine($P(t), \Theta_r, \Theta'_r$);
6 $P''(t) \leftarrow$ mutate($P'(t), \Theta_m, \Theta'_m$);
7 **Output:** $P''(t)$
8 **Input:** Θ'_s
9 $P(t+1) \leftarrow$ select($P''(t), \mu, \Theta_s, \Theta'_s$);
10 $t \leftarrow t+1$;
od

As in an automatic evolution system, there are parameters that are required to be fixed *a priori*: $\mu, \lambda, \Theta_\iota, \Theta_m, \Theta_r, \Theta_s$. There are, however, also parameters subject to change, $\Theta'_m, \Theta'_r, \Theta'_s$, depending on the user interaction with the IE system. Both parameter sets together determine the actual effect of mutation, recombination, and selection operators.

A simple variation of the standard algorithm shown above is to allow for population parameters to be also the subject of user interaction with the system. For example, some systems (Graf and Banzhaf 1995a) consider growing populations and a variable number of variants.

A more complicated variant of the standard algorithm would add a sorting process of variants according to a predefined fitness criterion. One step further is to allow this sorting process to result in a preselection in order to present a smaller number of variants for the interactive selection step. Both methods help the user to concentrate his or her selective action on the most promising variants according to this predefined criterion.

This algorithm is formulated as follows:

Input: $\mu, \lambda, \eta, \Theta_\iota, \Theta_\mathrm{m}, \Theta_\mathrm{o}, \Theta_\mathrm{r}, \Theta_\mathrm{s}$
Output: $\boldsymbol{a}^*$, the individual last selected during the run, or
P^*, the population last selected during the run.

```
1     t ← 0;
2     P(t) ← initialize(μ);
3     while (ι(P(t), Θ_ι) ≠ true) do
4           Input: Θ'_r, Θ'_m
5           P'(t)   ← recombine(P(t), Θ_r, Θ'_r);
6           P''(t)  ← mutate(P'(t), Θ_m, Θ'_m);
7           F(t)    ← evaluate(P''(t), λ);
8           P'''(t) ← sort(P''(t), Θ_o);
9           P''''(t) ← select(P'''(t), F(t), μ, η, Θ_s);
10          Output: P''''(t)
11          Input: Θ'_s
12          P(t + 1) ← select(P''''(t), μ, Θ_s, Θ'_s);
13          t ← t + 1;
      od
```

The newly added parameter Θ_o is used here to specify the predefined order of the result after evaluation according to the predefined criterion. As before, the Θ'_x-parameters are used to specify the user interaction with the system. η is the parameter stating how many of the automatically generated and ordered variants are to be presented to the user. If $\mu + \lambda = \eta$ in a $(\mu + \lambda)$-strategy, or $\lambda = \eta$ in a (μ, λ)-strategy, all variants will be presented for interactive selection. If, however, $\mu + \lambda > \eta$ and $\lambda > \eta$ respectively, solutions would be preselected and we speak of a hybrid evolution system (having elements of automatic as well as interactive evolution). Other parameters are used in the same way as in the standard algorithm.

C2.9.5 Difficulties

The second, more complicated version of IE requires a predefined fitness criterion, in addition to user action. This trades one advantage of IE systems for another: the absence of any requirement to quantify fitness for a small number of variants to be evaluated interactively by the user.

Interactive systems have one serious difficulty, especially in connection with the automatic means of variation that are usually provided: whereas the generation of variants does not necessarily require human intervention, selection of variants does call the attention of the user. Due to psychological constraints, however, humans can normally select only from a small set of choices. IE systems are thus constrained to present only of the order of ten choices at each point in time from which to choose. Also in sequence, only a limited number of generations can be practically inspected by a user before the user becomes tired.

It is emphasized that this limitation must not mean that the generation of variants has to be restricted to small numbers. Rather the variants have to be properly ordered at least, for a presentation of a subset that can be handled interactively.

C2.9.6 Application areas

An application of IE may be roughly divided into two parts:

(i) structural evolution by discrete combination of predefined elements and
(ii) parametric evolution of genes coding for quantifiable features of the phenotype.

All application use these parts to various degrees.

In the first part, one has to define the structure elements that might be combined into a correct genotype. Examples are symbolic expressions coding for appearance of points in an image plane (Sims 1991) or elementary geometric figures such as cone and cube (Todd and Latham 1992). In the second part, parameters have to be used to further specify features of these structural elements. Together, this information constitutes the genotype of the future design hopefully to be selected by a user. In a process called *expression* this genotype is then transformed into an image or three-dimensional form that can be displayed as a phenotype for the selection step.

Table C2.9.1 gives an overview of the presently used IE systems. The reader is advised to consult details with the sources given in the reference list.

Table C2.9.1. An overview of different IE systems.

Application	Genotypic elements	Phenotype	Source
Lifelike structures	line drawing parameters	biomorphs	Dawkins (1986)
Textures, images	math. functions, image processing op.	(x, y, z) pixel values	Sims (1991)
Animation	math. functions, image processing op.	(x, y, z) pixel values	Sims (1991)
Person tracking	(position of) facial parts	face images	Caldwell and Johnston (1991)
Images, sculptures	geometric forms and visually defined graphical elements	3D rendering of grown objects	Todd and Latham (1992)
Dynamical systems	CA rules, differential equations	system behavior	Sims (1992)
Images, animation	rules, parameters of L-systems	rendered objects	McCormack (1994)
Airplane design	structural elements, e.g. wings, body	airplane drawings	Nguyen and Huang (1994)
Images, design	tiepoints of bitmap images	bitmap images	Graf and Banzhaf (1995a)

Figure C2.9.1 illustrates some results with runs in different IE systems.

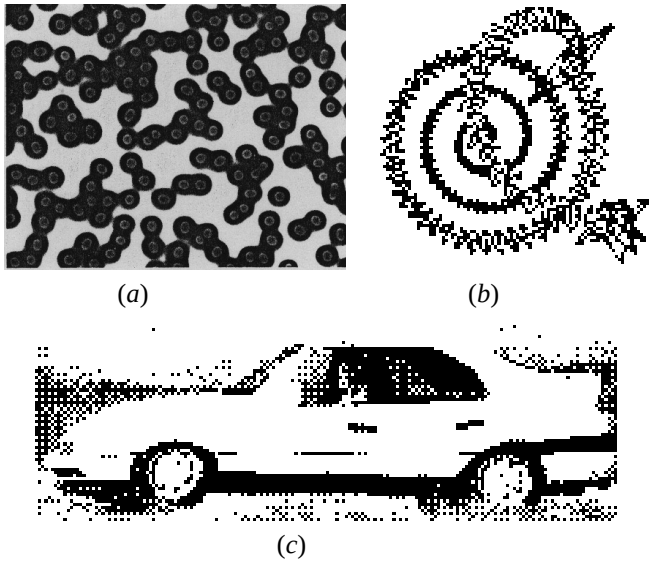


Figure C2.9.1. Samples of evolved objects: (*a*) dynamical system, cell structure (Sims 1992, copyright MIT Press); (*b*) artwork by Mutator (Todd and Latham 1992, with permission of the authors); (*c*) hybrid car model (Graf and Banzhaf 1995b, copyright IEEE Press).

Within the process of genotype–phenotype mapping a (recursive) developmental process is sometimes applied (Dawkins 1986, Todd and Latham 1992) whose results are finally displayed as the image for selection.

C2.9.7 Further developments and perspectives

As of now, the means to generate a small group of variants from which to choose interactively are still not very good. For example, one could imagine a tool for categorizing variants into a number of families of similar design and then present only one representative from each family. In this way, a large population of variants could be used in the background which is invisible to the user but might have beneficial effects in the course of evolution.

Another very interesting area of research is to assign *a posteriori* effective fitness values to members of the population, depending on user action. An individual which is selected more often would be assigned a higher fitness than an individual which is not. This might result in at least a crude measure of the

nonquantifiable fitness measures that lie at the heart of IE. One might even adjust the effect the operators have on the population, based on what is observed in the course of evolution directed by the user. In this way, an 'intelligent' system could be created, that is able to learn from actions of the user how to vary the population in order to arrive at good designs.

Another direction of research is to look into involving the user not (only) in the selection process, but in the variation process. Quite often, humans would have intuitive ideas for improvement of solutions when observing an automatic evolutionary process taking its steps. These ideas might be used to cut short the search routes an automatic algorithm is following. For this purpose, a user might be allowed to intervene in the process at appropriate interrupt times.

Finally, all sensory inputs could be used for IE. The systematic variation of components of a chemical compound that specifies an odor, for example, could be used to evolve a nice smelling perfume. Taste could as well be subject to interactive evolutionary tools, as could other objects if appropriately mapped to our senses (for instance by virtual reality tools).

With the advent of interactive media in the consumer market, production-on-demand systems might one day include an interactive evolutionary design device that allows the user not only to customize a product design before it goes into production, but also to generate his or her own original design that has never been realized before and usually will never be produced again. This would open up the possibility of evolutionary product design by companies which track their customers' activities and then distribute the best designs they discover.

References

Caldwell C and Johnston V S 1991 Tracking a criminal suspect through 'face-space' with a genetic algorithm *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, July 1991)* ed R K Belew and L B Booker (San Mateo, CA: Morgan Kaufmann) pp 416–21

Dawkins R 1986 *The Blind Watchmaker* (New York: Norton)

Graf J and Banzhaf W 1995a Interactive evolution of images *Proc. 4th Conf. on Evolutionary Programming (San Diego, CA, March 1995)* ed J R McDonnell, R G Reynolds and D Fogel (Cambridge: MIT Press) pp 53–65

——1995b An expansion operator for interactive evolution *Proc. 2nd IEEE Int. Conf. on Evolutionary Computation (Perth, November–December 1995)* (Piscataway, NJ: IEEE) pp 798–802

McCormack J 1994 Interactive evolution of L-system grammars for computer graphics modelling *Complex Systems* ed T Bossomaier and D Green (Singapore: World Scientific) pp 118–30

Nguyen T C and Huang T S 1994 Evolvable 3D modeling for model-based object recognition systems *Advances in Genetic Programming* ed K Kinnear (Cambridge: MIT Press) pp 459–75

Sims K 1991 Artificial evolution for computer graphics *Comput. Graph.* **25** 319–28

——1992 Interactive evolution of dynamical systems *Toward a Practice of Autonomous Systems* ed F J Varela and P Bourgine (Cambridge, MA: MIT Press) pp 171–8

Todd S and Latham W 1992 *Evolutionary Art and Computers* (London: Academic)

Further reading

This section is intended to give an overview of presently available work in IE and modeling methods which might be interesting to use.

1. Prusinkiewicz P and Lindenmayer A 1991 *The Algorithmic Beauty of Plants* (Berlin: Springer)

   An informative introduction to L-systems and their use in computer graphics.

2. Koza J R 1992 *Genetic Programming* (Cambridge, MA: MIT Press)

   A book describing methods to evolve computer code, mainly in the form of LISP-type S-expressions.

3. Caldwell C and Johnston V 1991 Tracking a criminal suspect through 'face-space' with a genetic algorithm *Proc. Int. Conf. on Genetic Algorithms (San Diego, CA, July 1991)* ed R K Belew and L B Booker (San Mateo, CA: Morgan Kaufmann) pp 416–21

   Very interesting work containing one of the more profane applications of IE. see al;so Section G8.3 of this handbook. G8.3

4. Baker E 1993 Evolving line drawings *Proc. Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) p 627

   This contribution discusses new ideas on design using simple style elements for IE.

5. Roston G P 1994 *A Genetic Methodology for Configuration Design* Doctoral Dissertation, Carnegie Mellon University

Informed discussion of different aspects of using genetic algorithms for design purposes.

C3.1 Introduction

Zbigniew Michalewicz

Abstract

This section provides a general introduction to search operators which have been proposed and investigated for various evolutionary computation techniques.

Any evolutionary system processes a population of individuals, $P(t) = \{\boldsymbol{a}_1^t, \ldots, \boldsymbol{a}_n^t\}$ (t is the iteration number), where each individual represents a potential solution to the problem at hand. As discussed in Chapter C1, many possible *representations* can be used for coding individuals; these representations may vary from binary strings to complex data structures I. C1

Each solution $\boldsymbol{a}_i^t$ is evaluated to give some measure of its *fitness*. Then a new population (iteration $t + 1$) is formed by selecting the more-fit individuals (the *selection* step of the evolutionary algorithm). Some members of the new population undergo transformations by means of 'genetic' operators to form new solutions. There are unary transformations m_i (mutation type), which create new individuals by a (usually small) change in a single individual ($m_i : I \rightarrow I$), and higher-order transformations c_j (crossover, or recombination type), which create new individuals by combining parts from several (two or more, up to the population size μ) individuals ($c_j : I^s \rightarrow I$, $2 \leq s \leq \mu$). C4 C2

It seems that, for any evolutionary computation technique, the representation of an individual in the population and the set of operators used to alter its genetic code constitute probably the two most important components of the system, and often determine the system's success or failure. Thus, a representation of object variables must be chosen along with the consideration of the evolutionary computation operators which are to be used in the simulation. Clearly, the reverse is also true: the operators of any evolutionary system must be chosen carefully in accordance with the selected representation of individuals. Because of this strong relationship between representations and operators, the latter are discussed with respect to some (standard) representations.

In general, Chapter C3 of the handbook provides a discussion on many operators which have been developed since the mid-1960s. Section C3.2 deals with mutation operators. Accordingly, several representations are considered (binary strings, real-valued vectors, permutations, finite-state machines, parse trees, and others) and for each representation one or more possible mutation operators are discussed. Clearly, it is impossible to provide a complete overview of *all* mutation operators, since the number of possible representations is unlimited. However, Section C3.2 provides a complete description of *standard* mutation operators which have been developed for *standard* data structures. Additionally, throughout the handbook the reader will find various mutations defined on other data structures (see, for example, Section G9.9 for a description of two problem-specific mutation operators which transform matrices). C3.2 G9.9

Section C3.3 deals with recombination operators. Again, as for mutation operators, several representations are considered (binary strings, real-valued vectors, permutations, finite-state machines, parse trees, and others) and for each representation several possible recombination operators are discussed. Recombination operators exchange information between individuals and are considered to be the main 'driving force' behind genetic algorithms, while playing no role in evolutionary programming. There are many important and interesting issues connected with recombination operators; these include properties that recombination operators should have to be useful (these are outlined by Radcliffe (1993)), the number of parents involved in recombination process (Eiben *et al* (1994) described experiments with multiparent recombination operators—so-called *orgies*), or the frequencies of recombination operators (these are discussed in Chapter E1 of the handbook together with heuristics for other parameter settings, C3.3 E1

such as population size or mutation).

Section C3.4 discusses some additional variations. These include the Baldwin effect, gene duplication and deletion, and knowledge-augmented operators. C3.4

References

Eiben A E, Raue P-E and Ruttkay Zs 1994 Genetic algorithms with multi-parent recombination *Proc. Parallel Problem Solving from Nature* vol 3 (New York: Springer) pp 78–87

Radcliffe N J 1993 Genetic set recombination *Foundations of Genetic Algorithms II (October 1994, Jerusalem)* ed Yu Davidor, H-P Schwefel and R Männer (San Mateo, CA: Morgan Kaufmann) pp 203–19

C3.2 Mutation

Thomas Bäck (C3.2.1), *David B Fogel* (C3.2.2, C3.2.4, C.3.2.6), *Darrell Whitley* (C3.2.3) *and Peter J Angeline* (C3.2.5, C3.2.6)

Abstract

See the individual abstracts for sections C3.2.1–C3.2.6.

C3.2.1 Binary strings

Thomas Bäck

Abstract

The mutation operator typically used in canonical genetic algorithms for fixed-length binary vectors is discussed in this section, presenting Holland's original definition of the operator as well as the standard realization as a bit inversion operator. An efficient implementation of the latter is outlined, and a brief overview of some common recommendations for setting the mutation rate p_m is given. It is argued that recent empirical and theoretical findings regarding the optimal mutation rate require a modification of the traditional interpretation of mutation in canonical genetic algorithms as a 'background operator' that serve only to support the crossover operator towards an interpretation of mutation as a powerful, constructive search operator on its own.

The mutation operator presently used in canonical *genetic algorithms* to manipulate binary vectors (also called *binary strings* or bitstrings) $\boldsymbol{a} = (a_1, \ldots, a_\ell) \in I = \{0,1\}^\ell$ of fixed length ℓ was originally introduced by Holland (1975, pp 109–11) for general finite individual spaces $I = A_1 \times \ldots \times A_\ell$, where $A_i = \{\alpha_{i_1}, \ldots, \alpha_{i_{k_i}}\}$. According to his definition, the mutation operator proceeds by: B1.2 C1.2

(i) determining the positions $i_1, \ldots, i_h$ ($i_j \in \{1, \ldots, \ell\}$) to undergo mutation by a uniform random choice, where each position has the same small probability p_m of undergoing mutation, independently of what happens at other positions, and

(ii) forming the new vector $\boldsymbol{a}' = (a_1, \ldots, a_{i_1-1}, a'_{i_1}, a_{i_1+1}, \ldots, a_{i_h-1}, a'_{i_h}, a_{i_h+1}, \ldots, a_\ell)$ where $a'_i \in A_i$ is drawn uniformly at random from the set of admissible values at position i.

The original value a_i at a position undergoing mutation is *not* excluded from the random choice of $a'_i \in A_i$; that is, although the position is chosen for mutation, the corresponding value might not change at all. This occurs with probability $1/|A_i|$, such that the effective (realized) mutation probability differs from p_m by a nonneglectible factor of 1/2 if a binary representation is used.

In order to avoid this problem, it is typically agreed on defining p_m to be the probability of independently inverting each of the variables $a_i \in \{0,1\}$, such that the mutation operator $m : \{0,1\}^\ell \to \{0,1\}^\ell$ produces a new individual $\boldsymbol{a}' = m(\boldsymbol{a})$ according to

$$a'_i = \begin{cases} a_i & u > p_m \\ 1 - a_i & u \leq p_m \end{cases} \tag{C3.2.1}$$

where $u \sim U([0,1))$ denotes a uniform random variable sampled anew for each $i \in \{1, \ldots, \ell\}$.

From a computational point of view, the straightforward implementation of equation (C3.2.1) as a loop calling the random number generator for each position i is extremely inefficient. Since the random variable T describing the distances between two positions to be mutated has a geometrical distribution with $\mathcal{P}\{T = t\} = p_m(1 - p_m)^{t-1}$ and expectation $\mathbf{E}[T] = 1/p_m$, and a geometrical random number can be generated according to

$$t = 1 + \left\lfloor \frac{\ln(1-u)}{\ln(1-p_m)} \right\rfloor \tag{C3.2.2}$$

(where $u \sim U([0, 1))$), equation (C3.2.2) provides an efficient method to generate the offset to find the next position for mutation from the current one. If the actual position plus the offset exceeds the vector dimension ℓ, it 'carries over' to the next individual and, if all individuals of the actual population have been processed, to the next generation (refer to Sections E2.1 and E2.2 for a more detailed discussion of the efficient implementation of this kind of mutation operator). E2.1, E2.2

Concerning the importance of mutation for the evolutionary search process, both Holland (1975, p 111) and Goldberg (1989, p 14) emphasize that mutation just serves as a 'background operator', supporting the *crossover* operator by assuring that the full range of allele values is accessible to the search. Consequently, quite small values of $p_m \in [0.001, 0.01]$ were recommended for canonical genetic algorithms (see e.g. De Jong 1975, Grefenstette 1986, Schaffer *et al* 1989) until recently, when both empirical and theoretical investigations clearly demonstrated the benefits of emphasizing the role of mutation as a search operator in these algorithms. More specifically, some of the important results include: C3.3.1

(i) empirical findings favoring an initially large mutation rate that exponentially decreases over time (Fogarty 1989),
(ii) the theoretical confirmation of the optimality of such an exponentially decreasing mutation rate for simple test functions (Hesser and Männer 1991, 1992, Bäck 1996), and
(iii) the knowledge of a lower bound $p_m = 1/\ell$ for the optimal mutation rate (Bremermann *et al* 1966, Mühlenbein 1992, Bäck 1993)

(see also Section E1.2 for a more detailed presentation of the corresponding mutation rate control policies). It is obvious from these results that not only for *evolution strategies* and *evolutionary programming*, but also for canonical genetic algorithms, mutation is an important search operator that cannot be neglected either in practical applications or in theoretical investigations of these algorithms. Moreover, it is also possible to release the user of a genetic algorithm from the problem of finding an appropriate mutation rate control or fine-tuning a fixed value by transferring the strategy parameter *self-adaptation* principle from evolution strategies and evolutionary programming to genetic algorithms (see Section C7.1 for a detailed presentation of existing approaches). E1.2, B1.3, B1.4, C7.1

C3.2.2 Real-valued vectors

David B Fogel

Abstract

There are a variety of methods to mutate real-valued vectors in evolutionary algorithms, and several such techniques are presented here.

Mutation generally refers to the creation of a new solution from one and only one parent (otherwise the creation is referred to as a *recombination*. Given a real-valued representation where each element in a population is an n-dimensional vector $x \in \mathbb{R}^n$, there are many methods for creating new elements (offspring) using mutation. These methods have a long history, extending back at least to Bremermann (1962), Bremermann *et al* (1965), and others. A variety of methods will be considered here. C3.3

The general form of mutation can be written as

$$\boldsymbol{x}' = m(\boldsymbol{x}) \tag{C3.2.3}$$

where $\boldsymbol{x}$ is the parent vector, m is the mutation function, and $\boldsymbol{x}'$ is the resulting offspring vector. Although there have been some attempts to include mutation operators that do not operate on the specific values of

the parents but instead simply choose $\boldsymbol{x}'$ from a fixed probability density function (PDF) (Montana and Davis 1989), such methods lose the inheritance from parent to offspring that can facilitate evolutionary optimization on a variety of response surfaces. The more common form of mutation generates an offspring vector:

$$\boldsymbol{x}' = \boldsymbol{x} + M \tag{C3.2.4}$$

where the mutation M is a random variable. M is often zero mean such that $E(\boldsymbol{x}') = \boldsymbol{x}$; the expected difference between a parent and its offspring is zero.

M can take different forms. For example, M could be the uniform random variable $U(a, b)^n$, where a and b are the lower and upper limits respectively. In this case, a is often set equal to $-b$. The result of applying this operator as M in (2) yields an offspring within a hyperbox $\boldsymbol{x} + U(-b, b)^n$. Although such a mutation is unbiased with respect to the position of the offspring within the hyperbox, the method suffers from easy entrapment when the parent vector $\boldsymbol{x}$ resides in a locally optimal well that is wider than the available step size. Davis (1989, 1991b) offered a similar operator (known as *creep*) that has a fixed probability of altering each component of $\boldsymbol{x}$ up or down by a bounded small random amount. The only method for alleviating entrapment in such cases relies on probabilistic selection, that is, maintaining a probability for choosing lesser-valued solutions to become parents of the subsequent generations (see Section C2.6). In contrast, unbounded mutation operators do not require such selection C2.6 methods to guarantee asymptotic global convergence (Fogel 1994, Rudolph 1994).

The primary unbounded mutation PDF for real-valued vectors has been the Gaussian (or 'normal') (Rechenberg 1973, Schwefel 1981, Fogel *et al* 1990, Fogel and Atmar 1990, Bäck and Schwefel 1993, Fogel and Stayton 1994, and many others). The PDF is defined as

$$g(x) = [\sigma(2\pi)^{1/2}]^{-1} \exp[-0.5(x-\mu)^2/\sigma^2].$$

When $\mu = 0$, the parameter σ offers the single control on the scaling of the PDF. It effectively generates a typical step size for a mutation. The use of zero-mean Gaussian mutations generates offspring that are (i) on average no different from their parents and (ii) increasingly less likely to be increasingly different from their parents. Saltations are not completely avoided such that any local optimum can be escaped from in a single iteration, yet they are not so common as to lose all inheritance from parent to offspring.

Other density functions with similar characteristics have also been implemented. Yao and Liu (1996) proposed using Cauchy distributions to aid in escaping from local minima (the Cauchy distribution has a fatter tail than the Gaussian) and demonstrated that Cauchy mutations may offer some advantages across a wide testbed of problems. Montana and Davis (1989) examined the use of Laplace-distributed mutations but there is no evidence that the Laplace distribution is particularly better suited than Gaussian or Cauchy mutations for typical real-valued optimization problems.

In the simplest version of evolution strategies or evolutionary programming, described as a $(1 + 1)$ evolutionary algorithm, a single parent $\boldsymbol{x}$ creates a single offspring $\boldsymbol{x}'$ by imposing a multivariate Gaussian perturbation with mean zero and standard deviation σ on the parent, then selects the better of the two trial solutions as the parent for the next iteration. The same standard deviation is applied to each component of the vector $\boldsymbol{x}$ during mutation. For some problems, the variation of σ (i.e. the step size control parameter in each dimension) can be computed to yield an optimal rate of convergence.

Let the convergence rate be defined as the ratio of the Euclidean distance covered toward the optimum solution to the number of trials required to achieve the improvement. Rechenberg (1973) calculated the convergence rates for two functions:

$$f_1(\boldsymbol{x}) = c_0 + c_1 x_1 \qquad i \in \{2, \ldots, n\} \quad -b/2 \leq x_i \leq b/2$$
$$f_2(\boldsymbol{x}) = \sum x_i^2$$

where $\boldsymbol{x} = (x_1, \ldots, x_n)^{\mathrm{T}} \in \mathbb{R}^n$. Function f_1 is termed the corridor model and represents a linear function with inequality constraints. Improvement is accomplished by moving along the first axis of the search space inside a corridor of width b. Function f_2 is termed the sphere model and is a simple n-dimensional quadratic bowl.

Rechenberg (1973) showed that the optimum rates of convergence (expected progress toward the optimum) are

$$\sigma = (\pi^{1/2}/2)(b/n)$$

on the corridor model, and

$$\sigma = 1.224\|\boldsymbol{x}\|/n$$

on the sphere model. That is, only a single step size control is needed for optimum convergence. Given these optimum standard deviations for mutation, the optimum probabilities of generating a successful mutation can be calculated as

$$p_1^{\text{opt}} = (2\text{e})^{-1} \approx 0.184$$
$$p_2^{\text{opt}} = 0.270.$$

Noting the similarity of these two values, Rechenberg (1973) proposed the following rule:

The ratio of successful mutations to all mutations should be $1/5$. If this ratio is greater than $1/5$, increase the variance; if it is less, decrease the variance.

Schwefel (1981) suggested measuring the success probability on-line over $10n$ trials (where there are n dimensions) and adjusting σ at iteration t by

$$\sigma(t) = \begin{cases} \sigma(t-n)\delta & \text{if } p_s < 0.2 \\ \sigma(t-n)\delta & \text{if } p_s > 0.2 \\ \sigma(t-n) & \text{if } p_s = 0.2 \end{cases}$$

with $\delta = 0.85$ and p_s equaling the number of successes in $10n$ trials divided by $10n$, which yields convergence rates of geometric order for both f_1 and f_2 (Bäck *et al* 1993; see the book by Bäck (1996) for corrections to the update rule offered by Bäck *et al* (1993)).

The use of a single step size control parameter covering all dimensions simultaneously is of limited robustness. The optimization performance can be improved by using appropriate step sizes in each dimension. This is particularly evident when consideration is given to optimizing a vector of parameters each of different units of dimension (e.g. temperature and pressure). Determining appropriate settings for each of n step sizes poses a significant challenge to the human operator; as such, methods have been proposed for self-adapting the step sizes concurrent to the evolutionary search.

The first efforts in self-adaptation date back at least to the article by Reed *et al* (1967), but the two most common implementations in use currently derive from the work of Schwefel (1981) and Fogel *et al* (1991). In each case, the vector of objective variables $\boldsymbol{x}$ is accompanied by a vector strategy parameters σ where σ_i denotes the standard deviation to use when applying a zero-mean Gaussian mutation to that component in the parent vector. The strategy parameters are updated by slightly different methods according to Schwefel (1981) and Fogel *et al* (1991).

Schwefel (1981) offered the procedure

$$\sigma_i' = \sigma_i \exp(\tau_0 N(0,1) + \tau N_i(0,1))$$
$$x_i' = x_i + N(0, \sigma_i')$$

where the constant $\tau \propto 1/[2(n^{1/2})]^{1/2}$, $\tau_0 \propto 1/(2n)^{1/2}$, $N(0,1)$ is a standard Gaussian random variable sampled once for all n dimensions and $N_i(0,1)$ is a standard Gaussian random variable sampled anew for each of the n dimensions. The procedure offers a general control for all dimensions and an individualized control for each dimension (Schwefel (1981) also offered a simplified method for self-adapting a single step size parameter σ). The values of σ' are, as shown, log-normal perturbations of their parent's vector σ.

Fogel *et al* (1991) independently offered the procedure

$$x_i' = x_i + N(0, \sigma_i)$$
$$\sigma_i' = \sigma_i + \chi N(0, \sigma_i)$$

where the parents' strategy parameters are used to create the offspring's objective values before being mutated themselves, and the mutation of the strategy parameters is achieved using a Gaussian distribution scaled by χ and the standard deviation for each dimension. This procedure also requires incorporating a rule such that if any component σ_i' becomes negative it is reset to an arbitrary small value ϵ.

Several comparisons have been conducted between these methods. Saravanan and Fogel (1994) and Saravanan *et al* (1995) indicated that the log-normal procedure offered by Schwefel (1981) generated

generally superior optimization performance (statistically significant) across a series of standard test functions. Angeline (1996a), in contrast, found that the use of Gaussian mutations on the strategy parameters generated better optimization performance when the objective function was made noisy. Gehlhaar and Fogel (1996) indicated that mutating the strategy parameters before creating the offspring objective values appears to be more generally useful both in optimizing a set of test functions and in molecular docking applications.

Both of the above methods for self-adaptation have been extended to include possible correlation across the dimensions. That is, rather than use n independent Gaussian random perturbations, a multivariate Gaussian mutation with arbitrary covariance can be applied. Schwefel (1981) described a method for incorporating rotation angles α such that new solutions are created by

$$\sigma_i' = \sigma_i \exp(\tau_0 N(0,1) + \tau N_i(0,1))$$
$$\alpha_j' = \alpha_j + \beta N_j(0,1)$$
$$x_i' = x_i + N(0, \sigma_i', \alpha_j')$$

where $\beta \approx 0.0873$ (5°), $i = 1, \ldots, n$ and $j = 1, \ldots, n(n-1)/2$, although it is not necessary to include all possible pairwise correlations in the method. Fogel *et al* (1992) offered a similar method operating directly on the components of the covariance matrix but the method does not guarantee positive definite matrices for $n > 2$, and the conventional method for implementing correlated mutation relies on the use of rotation angles as described above.

Another type of zero-mean mutation found in the literature is the so-called *nonuniform mutation* of Michalewicz (1996, pp 111–2), where

$$x_i'(t) = \begin{cases} x_i(t) + \Delta(t, \mathrm{ub}_i - x_i(t)) & \text{if } u < 0.5 \\ x_i(t) - \Delta(t, x_i(t) - \mathrm{lb}_i) & \text{if } u \geq 0.5 \end{cases}$$

where $x_i(t)$ is the ith parameter of the vector $\boldsymbol{x}$ at generation t, $x_i \in [\mathrm{lb}_i, \mathrm{ub}_i]$, the lower and upper bounds, respectively, u is a random uniform $U(0,1)$, and the function $\Delta(t, y)$ returns a value in the range $[0, y]$ such that the probability of $\Delta(t, y)$ being close to zero increases as t increases, essentially taking smaller steps on average. Michalewicz *et al* (1994) used the function

$$\Delta(t, y) = yu(1 - t/T)^b$$

where T is a maximal generation number and b is a system parameter chosen by the operator to determine the degree of nonuniformity.

There have been recent attempts to use nonzero-mean mutations on real-valued vectors. Ostermeier (1992) proposed an evolution strategy where the Gaussian mutations applied to the objective vector $\boldsymbol{x}$ are controlled by a vector of expectations μ as well as a vector of standard deviations σ. Ghozeil and Fogel (1996), following earlier work by Bremermann and Rogson (1964), have implemented a polar coordinate mutation in which new offspring are generated by perturbing the parent in a random direction (θ) with a specified step size (r).

C3.2.3 Permutations

Darrell Whitley

Abstract

Relatively few mutation operators have been defined for permutation based representations compared to the much larger number of recombination operators found in the literature. Some of the basic types of mutation operators are described, as well as related forms of well-known local search operators such as 2-opt.

C3.2.3.1 Introduction

Mutation operators can be used in a number of ways. Random mutation hillclimbing (Forrest and Mitchell 1993) is a search algorithm which applies a mutation operator to a single string and accepts any improving

moves. Some forms of evolutionary algorithms apply mutation operators to a population of strings without using recombination, while other algorithms may combine the use of mutation with recombination.

Any form of mutation which is to be applied to a permutation must yield a string which also represents a permutation. Most mutation operators for permutations are related to operators which have also been used in neighborhood local search strategies. Many of these operators thus can be applied in such as way that they reach a well-defined neighborhood of adjacent states.

*C3.2.3.2 2-opt, 3-opt, and **k**-opt*

The most common form of mutation is 2-opt (Lin and Kernighan 1973). Given a sequence of elements

A B C D E F G H

the 2-opt operator selects two points along the string, then reverses the segment between the points. Note that if the permutation is viewed as a circuit as in the *traveling salesman problem* (TSP), then all shifts of a sequence of N elements are equivalent. It follows that once two cut points have been selected in this circular string, it does not matter which segment is reversed; the effect is the same. G9.5

The 2-opt operator can be applied to all pairs of edges in $N(N-1)/2$ steps. This is analogous to one iteration of local search over all variables in a parameter optimization problem. If a full iteration of 2-opt to all pairs of edges fails to find an improving move, then a local optimum has been reached.

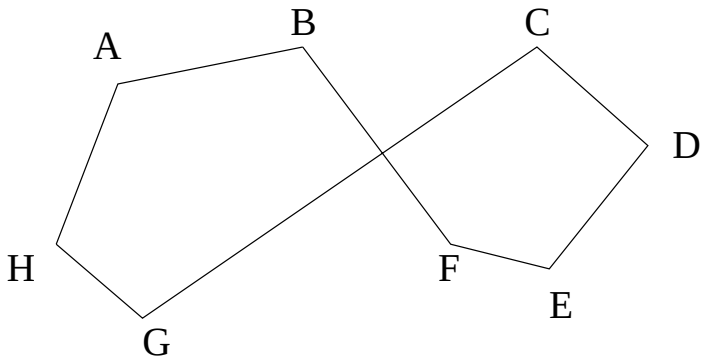


Figure C3.2.1. A graph.

2-opt is classically associated with the Euclidean TSP. Consider the graph in figure C3.2.1. If this is interpreted as a Euclidean TSP, then reversing the segment [C D E F] or the segment [G H A B] results in a graph where none of the edges cross and which has lower cost than the graph where the edges cross. Let {A, B, . . . , Z} be a set of vertices and (a, b) be the edge between vertices A and B. If vertices {B, C, F, G} in figure C3.2.1 are connected by the set of edges ((b, c), (b, f), (b, g), (c, f), (c, g) (f, g)), then two triangles are formed when B is connected to F and C is connected to G. To illustrate, create a new graph by placing a new vertex X at the point where the edges (b, f) and (c, g) cross. In the new graph in Euclidean space, the distance represented by edge (b, c) must be less than edges (b, x) + (x, c), assuming B, C, and X are not on a line; likewise, the distance represented by edge (f, g) must be less than edge (f, x) + (x, g). Thus, reversing the segment [C D E F] will always reduce the cost of the tour due to this triangle inequality. For the TSP this leads to the general principle that multiple applications of 2-opt will always yield a tour that has no crossed edges.

One can also look at reversing more than two segments at a time. The 3-opt operator cuts the permutation into three segments and then looks at all possible ways of reordering these segments. There are 3! = 6 ways to order the segments and each segment can be placed in a forward or reverse order. This yields up to $2^3 * 6 = 48$ possible new reorderings of the original permutation. For the symmetric TSP, however, all shifted arrangements of the three segments are equal and all reversed arrangements of the three segments are equal. Thus, the 3! orderings are all equivalent. (By analogy, note that there is only one possible Hamiltonian circuit tour between three cities.) This leaves only $2^3 = 8$ ways of placing each of the segments in a forward or reverse direction, each of which yields a unique tour. Thus, for

the symmetric TSP, the cost to test one 3-opt move is eight times greater than the cost of testing one 2-opt move. For other types of scheduling problem, such as resource allocation, reversals and shifts of the complete permutation are not necessarily equivalent and the cost of a 3-opt move may be up to 48 times greater than that of a 2-opt move. Also note that there are $\binom{N}{3}$ ways to break a permutation up into combinations of three segments compared to $\binom{N}{2}$ ways of breaking the permutation into two segments. Thus, the set of all possible 3-opt moves is much larger than the set of possible 2-opt moves. This further increases the cost of performing one pass of 3-opt over all possible ways of partitioning a permutation into three segments compared to a pass of 2-opt over all pairs of possible segments.

One can also use k-opt, where the permutation is broken into k segments, but such an operator will obviously be very costly.

C3.2.3.3 Insert, swap, and scramble operators

The TSP is sensitive to the adjacency of elements in a permutation, so that 2-opt represents a minimal change from one Hamiltonian circuit to another. For resource scheduling applications the permutation represent a priority queue and reversing a segment of a permutation represents a major change in access to available resources. For example, think of the permutation as representing a line of people waiting to buy a limited supply of tickets for different seats on different trains. The relative order of elements in the permutation tends to be important in this case and not the adjacency of the individual elements. In this case, a 2-opt segment reversal impacts many customers and is far from a minor change.

Radcliffe and Surry (1995) argue for representation-independent concepts of mutation and related forms of hillclimbers. Concerning desirable properties of a mutation operator, they state, 'One nearly universal characteristic, however, is that they ensure ... that the entire search space remains accessible from any population, and indeed from any individual. In most case mutation operators can actually move from any point in the search space to any other point directly, but the probability of making "large" moves is very much smaller than that of making "small" moves (at least with small mutation rates)' (p 58). They also suggest that a single mutation should represent a minimal change and look at different types of mutation operator for different representations of the TSP.

For resource allocation problems, a more modest change than 2-opt is to merely select one element and to *insert* it at some other position in the permutation. Syswerda (1991) refers to a variant of this as *position-based mutation* and describes it as selecting two elements and then moving the second element before the first element. Position-based mutation appears to be less general than the *insert* operator, since elements can only be moved forward in position-based mutation.

Similarly, one can select two elements and *swap* the positions of the two elements. Syswerda denotes this as *order-based mutation*. Note that if an element is moved forward or backward one position, this is equivalent to a swap of adjacent elements. One way in which swap can be used as a local search operator is to swap all adjacent elements, or perhaps also all pairs of elements. Finally, Syswerda also defines a *scramble* mutation operator that selects a sublist of permutation elements and randomly reorders (i.e. scrambles) the order of the subset while leaving the other elements in the permutation in the same absolute position. Davis (1991a) also reports on a scramble sublist mutation operator, except that the sublist is explicitly composed of contiguous elements of a permutation. (It is unclear whether Syswerda's scramble operator is also meant to work on contiguous elements or not; an operator that selects a sublist of elements over random positions of the permutation is certainly possible.)

For a problem that involved scheduling a limited number of flight simulators, Syswerda (1991, p 342) reported that when applied individually, the order-based swap mutation operator yielded the best results when compared to position-based mutation and scramble mutation. In this case the swaps were selected randomly rather than being performed over a fixed well-defined neighborhood. Davis (1991, p 81) on the other hand reports that the scramble sublist mutation operator proved to be better than the swap operator on a number of applications.

In conclusion, one cannot make *a priori* statements about the usefulness of a particular mutation operator without knowing something about the type of problem that is to be solved and the representation that is being used for that problem, but in general it is useful to distinguish between permutation problems that are sensitive to adjacency (e.g. the TSP) versus relative order (e.g. resource scheduling) or absolute position, which appears to be the least common.

C3.2.4 Finite-state machines

David B Fogel

Abstract

The form of a finite-state machine is given, along with typical procedure for mutating this structure in an evolutionary algorithm.

Given a *finite-state machine* representation where each element in a population is defined by a 5-tuple C1.5

$$M = (Q, T, P, s, o)$$

where Q is a finite set, the set of states, T is a finite set, the set of input symbols, P is a finite set, the set of output symbols, $s : Q \times T \rightarrow Q$, the next state function, and $o : Q \times T \rightarrow P$, the next output function, there are various methods for mutating parents to create offspring. Following directly from the definition, five obvious modes of mutation present themselves: (i) change an output symbol, (ii) change a state transition, (iii) add a new state, (iv) delete a state, and (v) change the start state. Each of these will be discussed in turn.

(i) Changing an output symbol consists of determining a particular state $q \in Q$, and then determining a particular symbol $\tau \in T$. For this pair (q, τ), identify the associated output symbol $\rho \in P$ and change it to a symbol chosen at random over the set P. The probability mass function for selecting a new symbol is typically uniform over the possible symbols in P, but can be chosen to reflect nearness between symbols or other known relationships between the symbols.

(ii) Changing a state transition consists of determining a particular state $q_1 \in Q$, and then determining a particular symbol $\tau \in T$. For this pair (q_1, τ), identify the associated next state q_2 and change it to a state chosen at random over the set Q. The probability mass function for selecting a new symbol is typically uniform over the possible states in Q.

(iii) Adding a state can only be performed when the maximum size of the machine has not been exceeded. The operation is accomplished by increasing the set Q by one element. This new state must be properly defined by generating an associated output symbol ρ_i and next state transition q_i for all input symbols $i = 1, \ldots, |T|$. The generation is typically performed by selecting output symbols and next state transitions with equal probability across their respective sets. Optionally, the new state may also be forced to be connected to the preexisting states by redirecting a randomly selected state transition of a randomly chosen preexisting state to the new state.

(iv) Deleting a state can be performed when the machine has at least two states. The operation is accomplished by decreasing the set Q by one element chosen at random (uniformly). All state transitions from other states that point to the deleted state must be redirected to the remaining states. This is often performed at random, with the new states selected with equal probability.

(v) Changing the start state can be performed when the machine has at least two states. The operation is accomplished by selecting a state $q \in Q$ to be the new starting state. Again, the selection is typically made uniformly over the available states.

The mutation operation can be implemented with various probabilities assigned to each mode of mutation (Fogel and Fogel 1986), although many of the initial experiments in evolutionary programming used equal probabilities (Fogel *et al* 1966). Further, multiple mutations can be performed (see e.g. Fogel *et al* 1966), and macromutations can be defined over pairs or higher-order combinations of these primitive operations. Recent efforts by Fogel *et al* (1994, 1995) and Angeline *et al* (1996) have incorporated the use of *self-adaptation* in mutating finite-state machines. C7.1

C3.2.5 Parse trees

Peter J Angeline

Abstract

Genetics-based evolutionary computations typically discount the role of mutation operation in the induction of evolved structures. This is especially true in genetic programming where mutation operations for parse trees are often not used. Some practitioners of genetic programming believe that mutation has an important role in evolving fit parse trees. This section describes several mutation operations for parse trees used by some genetic programming enthusiasts.

Standard *genetic programming* (Koza 1992), much as with traditional genetic algorithms, discounts mutation's role during evolution, often to an extreme (i.e. a mutation rate of zero). In many genetic programs, no mutation operations are used, which forces population sizes to be quite large in order to ensure access to all the primitives in the primitive language throughout a run. B1.5.1

In order to avoid unnecessarily large population sizes, Angeline (1996b) defines four distinct forms of mutation for *parse trees*. The *grow* mutation operator randomly selects a leaf from the tree and replaces it with a randomly generated new subtree (figure C3.2.2). The *shrink* mutation operator selects an internal node from the tree and replaces the subtree below it with a randomly generated leaf node (figure C3.2.3). The *switch* mutation operator selects an internal node from the parse tree and reorders its argument subtrees (figure C3.2.4). Finally, the *cycle* mutation operator selects a random node and replaces it with a new node of the same type (figure C3.2.5). If a leaf node is selected, then it is replaced by a leaf node. If an internal node is selected, then it is replaced by a function primitive that takes an equivalent number of arguments. Note that the mutation operation defined by Koza (1992) is a combination of a shrink mutation followed by a grow mutation at the same position. C1.6

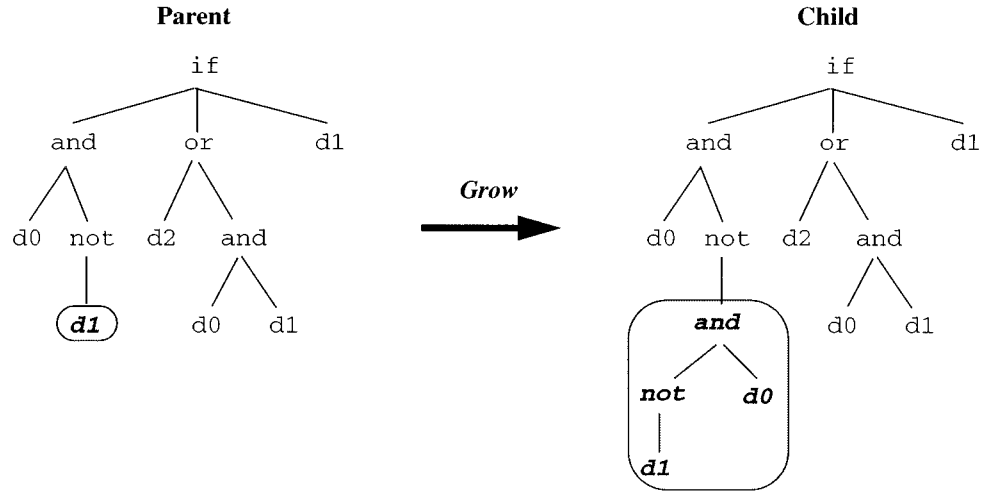


Figure C3.2.2. An illustration of the grow mutation operator applied to a Boolean parse tree. Given a parent tree to mutate, a terminal node is selected at random (highlighted) and replaced by a randomly generated subtree to produce the child tree.

Angeline (1996b) also defines a numerical terminal mutation that manipulates numerical terminals in a parse tree using the Gaussian mutations typically used in evolution strategies and evolutionary programming (see also Bäck 1996, Fogel 1995). In this mutation operation, a single numerical terminal in the parse tree is selected at random and a Gaussian random variable with a user-defined variance is added to its value.

If the application of a mutation operation creates a parse tree that violates the size limitation criteria for the parse tree, typically the operation is revoked and the state of the parse tree prior to the operation is restored. In some cases, when a series of mutations are to be performed, as in Angeline (1996b), the complete set of mutations is executed prior to checking whether the mutated parse tree conforms to the imposed size restrictions.

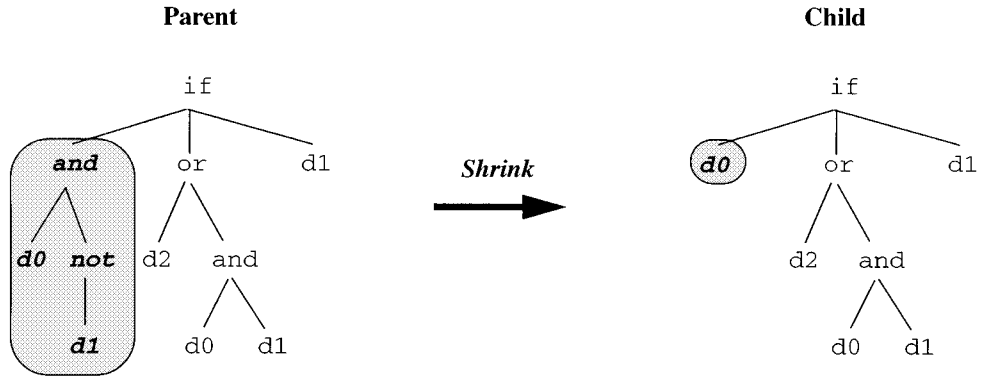


Figure C3.2.3. An illustration of the shrink mutation operator applied to a Boolean parse tree. Given a parent tree to mutate, an internal function node is selected at random (highlighted) and replaced by a randomly selected terminal to produce the child tree.

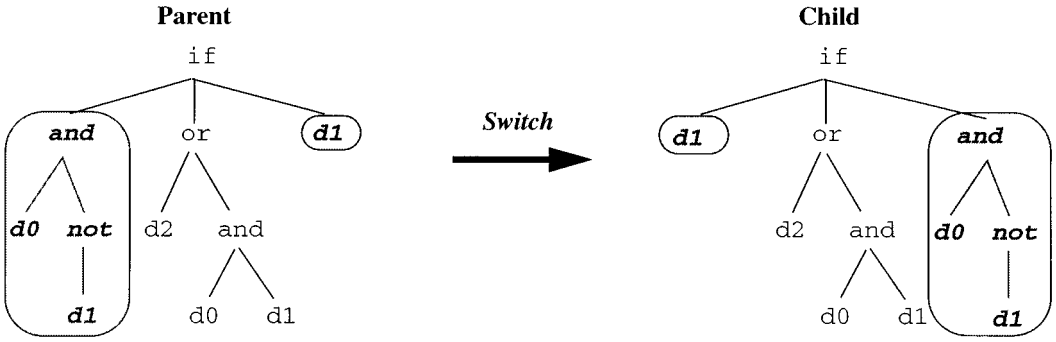


Figure C3.2.4. An illustration of the switch mutation operator applied to a Boolean parse tree. Given a parent tree to mutate, an internal function node is selected, two of the subtrees below it are selected (highlighted in the figure) and their positions switched to produce the child tree.

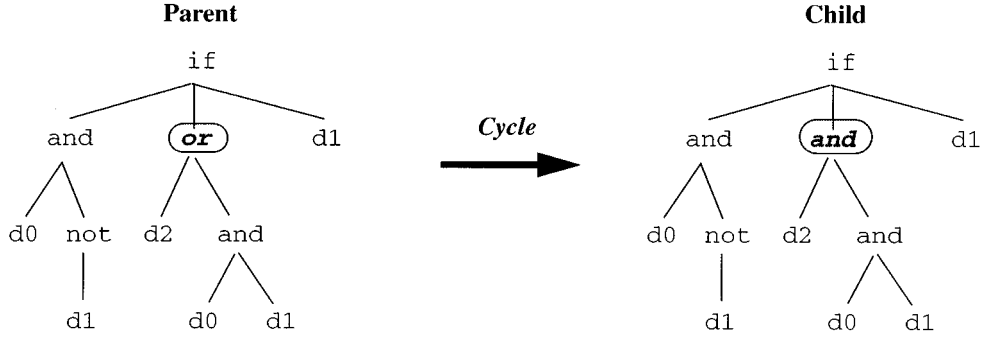


Figure C3.2.5. An illustration of the cycle mutation operator applied to a Boolean parse tree. Given a parent tree to mutate, a single node, either a terminal or function, is selected at random (highlighted in the parent) and replaced by a randomly selected node with the same number of arguments to produce the child tree.

When evolving typed parse trees as in Montana (1995), mutation must also be sensitive to the return type of the manipulated node. In order to preserve the syntactic constraints, the return type of the node after mutation must be the same. This is accomplished by keeping track of the return types for the various primitives in the language and restricting mutation to return those primitives with the corresponding type.

C3.2.6 Other representations

David B Fogel and Peter J Angeline

Abstract

We consider the mutation of hybrid representations, including mixed-integer representations. Also discussed are data structures incorporating introns.

Many real-world applications suggest the use of representations that are hybrids of the canonical representations. One common instance is the simultaneous use of discrete and continuous object variables, with a general formulation of the global optimization problem as follows (Bäck and Schütz 1995):

$$\min\{f(\boldsymbol{x}, \boldsymbol{d}) | \boldsymbol{x} \in M, R^n \supseteq M, \boldsymbol{d} \in N, Z^{n_d} \supseteq N\}.$$

Within evolution strategies and evolutionary programming, the common representation is simply the real-integer vector pair (i.e. no effort is made to encode these vectors into another representation such as binary).

The simple approach to mutating such a representation would be to embed the integers in the real numbers and use the standard methods of mutation (e.g. Gaussian random perturbation) found in evolution strategies and evolutionary programming. The results could be rounded to the integers when dealing with the elements in $\boldsymbol{d}$. Bäck and Schütz (1995) note, however, that, for a discrete optimization problem, the 'optimum point obtained by rounding the results of the continuous optimization might be different from the true discrete optimum point even for linear objective functions with linear constraints'. Bäck and Schütz (1995) also note the potential problems in optimizing $\boldsymbol{x}$ and $\boldsymbol{d}$ separately (as in the work of Lohmann (1992) and Fogel (1991, 1993) among others) because there may be interdependences between the appropriate mutations to $\boldsymbol{x}$ and $\boldsymbol{d}$.

Bäck and Schütz (1995) approach the general problem by including a vector of mutation strategy parameters $p_j \in (0, 1)$ and $j = 1, 2, \ldots, d$, where there are d components to the vector $\boldsymbol{d}$. (Alternatively, fewer strategy parameters could be used.) These strategy parameters are adapted along with the usual step size control strategy parameters for Gaussian mutation of the real-world vector $\boldsymbol{x}$. The discrete strategy parameters are updated by the formula

$$p'_j = \left(\frac{1 + (1 - p_j)}{p_j \times \exp[-\gamma N_j(0, 1)]} \right)^{-1}$$

where γ is set proportional to $[2(d)^{1/2}]^{-1/2}$. Actual mutation to the parameters in $\boldsymbol{d}$ can be accomplished using an appropriate random variable (e.g. uniform or Poisson).

With regard to mutation in introns, because the introns are not coded into functional behavior (i.e. they do not affect performance in terms of the objective function), the manner in which they are mutated is irrelevant.

In the standard genetic algorithm representation, the semantics of an allele value (how the allele is interpreted) are typically tied to its position in the fixed-length n-ary string. For instance, in a binary string representation, each position signifies the presence or absence of a specific feature in the genome being decoded. The difficulty with such a representation is that with positions in the string representation that are semantically linked but separated by a large number of intervening positions in the string, crossover has a high probability of disrupting beneficial settings for these two positions. Goldberg *et al* (1989) describe a representation for a genetic algorithm that embodies one approach to addressing this problem. In their *messy genetic algorithm* (mGA), each allele value is represented as a pair of values, one specifying the actual allele value and one specifying the position the allele occupies. Messy GAs are defined to be of variable length, and Goldberg *et al* (1989) describe appropriate methods for resolving underdetermined or overdetermined genomes. In this representation it is important to note that the semantics are literally carried along with the allele value in the form of the allele's string position.

C4.2.4

Diplodic representations, representations that include multiple allele values for each position in the genome, have been offered as mechanisms for modeling cyclic environments. In a diplodic representation, a method for determining which allele value for a gene will be expressed is required to adjudicate when the allele values do not agree. Building on earlier investigations (e.g. Bagley 1967, Hollstein 1971, Brindle

1981) Goldberg and Smith (1987) demonstrate that an evolving dominance map allows quicker adaptation to cyclical environment changes than either a haploid representation or a diploid representation using a fixed dominance mapping. In the article by Goldberg and Smith (1987), a triallelic representation from the dissertation of Hollstein (1971) is used: 1, i, and 0. Both 1 and i map to the allele value of '1', while 0 maps to the allele value of '0' with 1 dominating both i and 0 and 0 dominating i. Thus, the dominance of a 1 over a 0 allele value could be altered via mutation by altering the value to an i. Ng and Wong (1995) extend the multiallele approach to dominance computation by adding a fourth value for a recessive 0. Thus 1 dominates 0 and o while 0 dominates i and o. When both allele values for a gene are dominant or recessive, then one of the two values is chosen randomly to be the dominant value. Ng and Wong (1995) also suggest that the dominance of all of the components in the genome should be reversed when the fitness value of an individual falls by 20% or more between generations.

References

Angeline P J 1996a The effects of noise on self-adaptive evolutionary optimization, *Proc. 5th Ann. Conf. on Evolutionary Programming* ed L J Fogel, P J Angeline and T Bäck (Cambridge, MA: MIT Press) pp 443–50

——1996b Genetic programming's continued evolution *Advances in Genetic Programming* vol 2, ed P J Angeline and K Kinnear (Cambridge, MA: MIT Press) pp 89–110

Angeline P J, Fogel D B and Fogel L J 1996 A comparison of self-adaptation methods for finite state machines in a dynamic environment *Proc. 5th Ann. Conf. on Evolutionary Programming* ed L J Fogel, P J Angeline and T Bäck (Cambridge, MA: MIT Press) pp 441–50

Bäck T 1993 Optimal mutation rates in genetic search *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 2–8

——1996 *Evolutionary Algorithms in Theory and Practice* (New York: Oxford University Press)

Bäck T, Rudolph G and Schwefel H-P 1993 Evolutionary programming and evolution strategies: similarities and differences *Proc. 2nd Ann. Conf. on Evolutionary Programming (San Diego, CA)* ed D B Fogel and W Atmar (La Jolla, CA: Evolutionary Programming Society) pp 11–22

Bäck T and Schütz M 1995 Evolution strategies for mixed-integer optimization of optical multilayer systems *Proc. 4th Ann. Conf. on Evolutionary Programming (San Diego, CA, March 1995)* ed J R McDonnell, R G Reynolds and D B Fogel (Cambridge, MA: MIT Press) pp 33–51

Bäck T and Schwefel H-P 1993 An overview of evolutionary algorithms for parameter optimization *Evolutionary Comput.* **1** 1–24

Bagley J D 1967 *The Behavior of Adaptive Systems which Employ Genetic and Correlation Algorithms* Doctoral Dissertation, University of Michigan; University Microfilms 68-7556

Bremermann H J 1962 Optimization through evolution and recombination *Self-Organizing Systems* ed M C Yovits, G T Jacobi and G D Goldstine (Washington, DC: Spartan) pp 93–106

Bremermann H J and Rogson M 1964 *An Evolution-type Search Method for Convex Sets* ONR Technical Report, contracts 222(85) and 3656(58)

Bremermann H J, Rogson M and Salaff S 1965 Search by evolution *Biophysics and Cybernetic Systems* ed M Maxfield, A Callahan and L J Fogel (Washington, DC: Spartan) pp 157–67

——1966 Global properties of evolution processes *Natural Automata and Useful Simulations* ed H H Pattec, E A Edelsack, L Fein and A B Callahan (Washington, DC: Spartan) pp 3–41

Brindle A 1981 *Genetic Algorithms for Function Optimization* Doctoral Dissertation, University of Alberta

Davis L 1989 Adapting operator probabilities in genetic algorithms *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, June 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 61–9

Davis L 1991a *Handbook of Genetic Algorithms* (New York: Van Nostrand Reinhold)

——1991b A genetic algorithms tutorial *Handbook of Genetic Algorithms* ed L Davis (New York: Van Nostrand Reinhold) pp 1–101

De Jong K A 1975 *An Analysis of the Behaviour of a Class of Genetic Adaptive Systems* PhD Thesis, University of Michigan

Fogarty T C 1989 Varying the probability of mutation in the genetic algorithm *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 104–9

Fogel D B 1991 *System Identification through Simulated Evolution* (Needham, MA: Ginn)

——1993 Using evolutionary programming to construct neural networks that are capable of playing tic-tac-toe *Proc. 1993 IEEE Int. Conf. on Neural Networks* (Piscataway, NJ: IEEE) pp 875–80

——1994 Asymptotic convergence properties of genetic algorithms and evolutionary programming: analysis and experiments *Cybern. Syst.* **25** 389–407

——1995 *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence* (New York: IEEE)

Fogel D B and Atmar J W 1990 Comparing genetic operators with Gaussian mutations in simulated evolutionary processes using linear systems *Biol. Cybern.* **63** 111–4

Fogel D B, Fogel L J and Atmar J W 1991 Meta-evolutionary programming *Proc. 25th Asilomar Conf. on Signals, Systems, and Computers* ed R R Chen (Pacific Grove, CA: Maple) pp 540–5

Fogel D B, Fogel L J, Atmar W and Fogel G B 1992 Hierarchic methods of evolutionary programming *Proc. 1st Ann. Conf. on Evolutionary Programming* ed D B Fogel and W Atmar (La Jolla, CA: Evolutionary Programming Society) pp 175–82

Fogel D B and Stayton L C 1994 On the effectiveness of crossover in simulated evolutionary optimization *BioSystems* **32** 171–82

Fogel L J, Angeline P J and Fogel D B 1994 A preliminary investigation on extending evolutionary programming to include self-adaptation on finite state machines *Informatica* **18** 387–98

——1995 An evolutionary programming approach to self-adaptation in finite state machines *Proc. 4th Ann. Conf. on Evolutionary Programming (San Diego, CA, March, 1995)* ed J R McDonnell, R G Reynolds and D B Fogel (Cambridge, MA: MIT Press) pp 355–65

Fogel L J and Fogel D B 1986 *Artificial Intelligence through Evolutionary Programming* Final Report for US Army Research Institute, contract no PO-9-X56-1102C-1

Fogel L J, Owens A J and Walsh M J 1966 *Artificial Intelligence Through Simulated Evolution* (New York: Wiley)

Forrest S and Mitchell M 1993 Relative building-block fitness and the building block hypothesis *Foundations of Genetic Algorithms 2* ed D Whitley (San Mateo, CA: Morgan Kaufmann) pp 109–26

Gehlhaar D K and Fogel D B 1996 Tuning evolutionary programming for conformationally flexible molecular docking *Proc. 5th Ann. Conf. on Evolutionary Programming* ed L J Fogel, P J Angeline and T Bäck (Cambridge, MA: MIT Press) at press

Ghozeil A and Fogel D B 1996 A preliminary investigation into directed mutations in evolutionary algorithms *Parallel Problem Solving from Nature 4* to appear

Goldberg D E 1989 *Genetic Algorithms in Search, Optimization and Machine Learning* (Reading, MA: Addison-Wesley)

Goldberg D E, Korb D E and Deb K 1989 Messy genetic algorithms: motivation, analysis, and first results *Complex Syst.* **3** 493–530

Goldberg D E and Smith R E 1987 Nonstationary function optimization using genetic algorithms with dominance and diploidy *Proc. 2nd Int. Conf. on Genetic Algorithms (Cambridge, MA, 1987)* ed J J Grefenstette (Hillsdale, NJ: Erlbaum) pp 59–68

Grefenstette J J 1986 Optimization of control parameters for genetic algorithms *IEEE Trans. Syst. Man Cybernet.* **SMC-16** 122–8

Hesser J and R Männer 1991 Towards an optimal mutation probability in genetic algorithms *Proc. 1st Conf. on Parallel Problem Solving from Nature (Dortmund, 1990) (Lecture Notes in Computer Science 496)* ed H-P Schwefel and R Männer (Berlin: Springer) pp 23–32

——1992 Investigation of the m-heuristic for optimal mutation probabilities *Parallel Problem Solving from Nature, 2 (Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature, Brussels, 1992)* ed R Männer and B Manderick (Amsterdam: Elsevier) pp 115–24

Holland J H 1975 *Adaptation in Natural and Artificial Systems* (Ann Arbor, MI: University of Michigan Press)

Hollstein R B 1971 *Artificial Genetic Adaptation in Computer Control Systems* Doctoral Dissertation, University of Michigan; University Microfilms 71-23, 773

Koza J R 1992 *Genetic Programming: On the Programming of Computers by Means of Natural Selection* (Cambridge, MA: MIT Press)

Lin S and Kernighan B 1973 An efficient heuristic procedure for the traveling salesman problem *Operations Res.* **21** 498–516

Lohmann, R 1992 Structure evolution in neural systems *Dynamic, Genetic, and Chaotic Programming* ed B Soucek (New York: Wiley) pp 395–411

Michalewicz Z 1996 *Genetic Algorithms + Data Structures = Evolution Programs* 3rd edn (Berlin: Springer)

Michalewicz Z, Logan T and Swaminathan S 1994 Evolutionary operators for continuous convex parameter spaces *Proc. 3rd Ann. Conf. on Evolutionary Programming (San Diego, CA, February 1994)* ed A V Sebald and L J Fogel (Singapore: World Scientific) pp 84–97

Montana D J 1995 Strongly typed genetic programming *Evolutionary Comput.* **3** 199–230

Montana D J and Davis L 1989 Training feedforward neural networks using genetic algorithms *Proc. 11th Int. Joint Conf. on Artificial Intelligence* ed N S Sridharan (San Mateo, CA: Morgan Kaufmann) pp 762–7

Mühlenbein H 1992 How genetic algorithms really work: I mutation and hillclimbing *Parallel Problem Solving from Nature, 2 (Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature, Brussels, 1992)* ed R Männer and B Manderick pp 15–25

Ng K P and Wong K C 1995 A new diploid scheme and dominance change mechanism for non-stationary function optimization *Proc. 6th Int. Conf. on Genetic Algorithms (Pittsburgh, PA, July 1995)* ed L J Eshelman (San Mateo, CA: Morgan Kaufmann) pp 159–66

Ostermeier A 1992 An evolution strategy with momentum adaptation of the random number distribution *Parallel Problem Solving from Nature, 2 (Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature, Brussels, 1992)* ed R Männer and B Manderick pp 197–206

Radcliffe N and Surry P D 1995 Fitness variance of formae and performance prediction *Foundations of Genetic Algorithms 3* ed D Whitley and M Vose (San Mateo, CA: Morgan Kaufmann) pp 51–72

Rechenberg I 1973 *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution* (Stuttgart: Frommann-Holzboog)

Reed J, Toombs R and Barricelli N A 1967 Simulation of biological evolution and machine learning *J. Theor. Biol.* **17** 319–42

Rudolph G 1994 Convergence properties of canonical genetic algorithms *IEEE Trans. Neural Networks* **5** 96–101

Saravanan N and Fogel D B 1994 Learning strategy parameters in evolutionary programming: an empirical study *Proc. 3rd Ann. Conf. on Evolutionary Programming (San Diego, CA, February 1994)* ed A V Sebald and L J Fogel (Singapore: World Scientific) pp 269–80

Saravanan N, Fogel D B and Nelson K M 1995 A comparison of methods for self-adaptation in evolutionary algorithms *BioSystems* **36** 157–66

Schwefel H-P 1981 *Numerical Optimization of Computer Models* (Chichester: Wiley)

——1995 *Evolution and Optimum Seeking* (New York: Wiley)

Schaffer J D, Caruana R A, Eshelman L J and Das R 1989 A study of control parameters affecting online performance of genetic algorithms for function optimization *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, June 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 51–60

Syswerda G 1991 Schedule optimization using genetic algorithms *Handbook of Genetic Algorithms* ed L Davis (New York: Van Nostrand Reinhold) pp 332–49

Yao X and Liu Y 1996 Fast evolutionary programming *Proc. 5th Ann. Conf. on Evolutionary Programming* ed L J Fogel, P J Angeline and T Bäck (Cambridge, MA: MIT Press) at press

C3.3 Recombination

Lashon B Booker (C3.3.1), *David B Fogel* (C3.3.2, C3.3.4, C3.3.6), *Darrell Whitley* (C3.3.3) *and Peter J Angeline* (C3.3.5, C3.3.6)

Abstract

See the individual abstracts for sections C3.3.1–C3.3.6

C3.3.1 Binary strings

Lashon B Booker

Abstract

We describe various approaches to implementing crossover operators for recombining linear strings. The discussion includes an explanation of the principal mechanisms underlying the most widely used crossover operators. An overview of existing formal analyses of crossover is also provided, followed by a brief discussion of the search biases associated with crossover.

C3.3.1.1 Introduction

In *biological systems*, crossing-over is a complex process that occurs between pairs of chromosomes. A2.2.4 Two chromosomes are physically aligned, breakage occurs at one or more corresponding locations on each chromosome, and homologous chromosome fragments are exchanged before the breaks are repaired. This results in a recombination of genetic material that contributes to variability in the population. In evolutionary algorithms, this process has been abstracted into syntactic crossing-over (or crossover) operators that exchange substrings between chromosomes represented as linear strings of symbols. In this section we describe various approaches to implementing these computational recombination techniques. Note that, while *binary strings* are the canonical representation of chromosomes most often associated C1.2 with evolutionary algorithms, crossover operators work the same way on all linear strings regardless of the cardinality of the symbol alphabet. Accordingly, the discussion in this section applies to both binary and nonbinary string representations. The obvious caveat is that the syntactic manipulations by crossover must yield semantically valid results. When this becomes a problem—for example, when the chromosomes represent permutations (see Section C1.4)—then other syntactic operations must be used. C1.4

C3.3.1.2 Principal mechanisms

The basic crossover operation, introduced by Holland (1975), is a three-step procedure. First, two individuals are chosen at random from the population of 'parent' strings generated by the selection (see Chapter C2) operator. Second, one or more string locations are chosen as breakpoints (or *crossover points*) C2 delineating the string segments to exchange. Finally, parent string segments are exchanged and then combined to produce two resultant 'offspring' individuals. The proportion of parent strings undergoing crossover during a generation is controlled by the *crossover rate*, $p_c \in [0, 1]$, which determines how frequently the crossover operator is invoked. Holland illustrates how to implement this general procedure

by describing the simple one-point crossover operator. Given parent strings $\boldsymbol{x}$ and $\boldsymbol{y}$, a crossover point is selected by randomly choosing an integer $k \sim U(1, \ell - 1)$:

$$\begin{array}{l}(x_1 \ldots x_k x_{k+1} \ldots x_\ell) \\ (y_1 \ldots y_k y_{k+1} \ldots y_\ell)\end{array} \Longrightarrow \begin{array}{l}(x_1 \ldots x_k y_{k+1} \ldots y_\ell) \\ (y_1 \ldots y_k x_{k+1} \ldots x_\ell).\end{array}$$

Two new resultant strings are formed by exchanging the parent substrings to the right of position k. Holland points out that when the overall algorithm is limited to producing only one new individual per generation, one of the resultant strings generated by this crossover operator must be discarded. The discarded string is usually chosen at random.

Holland's general procedure defines a family of operators that can be described more formally as follows. Given a space I of individual strings, a crossover operator is a mapping

$$r : I \times I \xrightarrow{\boldsymbol{m}} I \times I \qquad r(\boldsymbol{a}, \boldsymbol{b}) = (\boldsymbol{c}, \boldsymbol{d})$$

where $\boldsymbol{m} \in \mathbf{B}^\ell$ and

$$c_i = \begin{cases} a_i & \text{if } m_i = 0 \\ b_i & \text{if } m_i = 1 \end{cases} \qquad d_i = \begin{cases} b_i & \text{if } m_i = 0 \\ a_i & \text{if } m_i = 1. \end{cases}$$

Although this formal description characterizes crossover as a binary operator, there are some implementations of crossover involving more than two parents (e.g. the multiparent uniform crossover operator described by Furuya and Haftka (1993) and the scanning crossover and diagonal crossover operators described by Eiben *et al* (1995)).

The binary string $\boldsymbol{m}$ is a *mask* computed for each invocation of the operator from the set of crossover points. This mask identifies which string segments will be exchanged during the crossover operation. Note that the mask $\boldsymbol{m}$ and its complement $\mathbf{1} - \boldsymbol{m} = (1 - m_1 \ldots 1 - m_\ell)$ generate the same (unordered) set of resultant strings. Another way to interpret the mask is as a specification of which parent provided the symbol at each position in a resultant string. A crossover operation can be viewed as the simultaneous occurrence of two *recombination events*, each producing one of the two offspring. The pair $(\boldsymbol{m}, \mathbf{1} - \boldsymbol{m})$ can be used to designate these recombination events. Each symbol in a resultant string is either transmitted by the first parent (denoted in the mask by zero) or the second parent (denoted by one). Consequently, the event generating string $\boldsymbol{c}$ above is specified by $\boldsymbol{m}$ and the event generating $\boldsymbol{d}$ is specified by $\mathbf{1} - \boldsymbol{m}$.

A simple pseudocode description of how to implement one of these crossover operators is given below:

```
crossover(a, b) :
    sample u ∈ U(0, 1)
    if (u > p_c)
    then return(a, b)
    fi
    c := a;
    d := b;
    m := compute_mask();
    for i := 1 to ℓ do
        if (m_i = 1)
        then
            c_i := b_i;
            d_i := a_i;
        fi
    od
    return(c, d);
```

Empirical studies have shown that the best setting for the crossover rate p_c depends on the choices made regarding other aspects of the overall algorithm, such as the settings for other parameters such as population size and mutation rate, and the selection operator used. Some commonly used crossover rates are $p_c = 0.6$ (De Jong 1975), $p_c \in [0.45, 0.95]$ (Grefenstette 1986), and $p_c \in [0.75, 0.95]$ (Schaffer *et al* 1989). Techniques for adaptively modifying the crossover rate have also proven to be useful (Booker 1987, Davis 1989, Srinivas and Patnaik 1994, Julstrom 1995).

The pseudocode shown above makes it clear that the differences between crossover operators are most likely to be found in the implementation of the compute_mask() procedure. The following examples of pseudocode characterize the way compute_mask() is implemented for the most commonly cited crossover operators.

One-point crossover. A single crossover point is selected. This operator can only exchange contiguous substrings that begin or end at the endpoints of the chromosome. This is rarely used in practice.

```
sample u ∈ U(1, ℓ − 1)
m := 0;
for i := u + 1 to ℓ do
    m_i = 1;
od
return m;
```

n-point crossover. This operator, first implemented by De Jong (1975), generalizes one-point crossover by making the number of crossover points a parameter. The value $n = 2$ designating two-point crossover is the choice that minimizes disruptive effects (see the discussion of disruption in section C3.3.1.3) and is frequently used in applications. There is no consensus about the advantages and disadvantages of using values $n \geq 3$. Empirical studies on this issue (De Jong 1975, Eshelman *et al* 1989) are inconclusive.

```
sample u_1, ..., u_n ∈ U(1, ℓ), u_1 ≤ ··· ≤ u_n
if ((n mod 2) = 1)
then u_{n+1} := ℓ;
fi
m := 0;
for j := 1 to n step 2 do
    for i := u_j + 1 to u_{j+1} do
        m_i = 1;
    od
od
return m;
```

By convention (De Jong 1975), when n is odd an additional crossover point is assumed to occur at position ℓ. Note that many implementations select the crossover points without replacement—instead of with replacement as indicated here—to guarantee that the crossover points are distinct. Analysis of disruptive effects has shown that there are only small differences in the two approaches (see the discussion of disruption in section C3.3.1.3) and no empirical differences in performance have been reported.

Uniform crossover. This is an operator introduced by Ackley (1987) but most often attributed to Syswerda (1989). (The basic idea can be traced to early work in mathematical population genetics, see Geiringer (1944)). The number of crossover points is not fixed in advance. Instead, the decision to insert a breakpoint is made independently at each string position. This operator is frequently used in applications.

```
m := 0;
for i := 1 to ℓ do
    sample u ∈ U(0, 1)
    if (u ≤ p_x)
    then m_i = 1;
    fi
od
return m
```

The value $p_x = 0.5$ first used by Ackley remains the standard setting for the crossover probability at each position, though it may be advantageous to use smaller values (Spears and De Jong 1991b). When $p_x = 0.5$, every binary string of length ℓ is equally likely to be generated as a mask. In this case, it is often more efficient to implement the operator by using a random integer sampled from $U(0, 2^\ell - 1)$ as the mask instead of constructing the mask one bit at a time.

Punctuated crossover. Rather than computing the crossover mask directly, Schaffer and Morishima (1987) used a binary string of 'punctuation marks' to indicate the location of crossover points for a multipoint crossover operation. The extra information was appended to the chromosome so that the number and location of crossover points could be manipulated by genetic search. The resulting representation used by the punctuated crossover operator is a string of length 2ℓ, $\boldsymbol{x} = (x_1 \ldots x_\ell x'_1 \ldots x'_\ell)$, where x_i is the symbol at position i and x'_i is a punctuation mark that is 1 if position i is a crossover point and 0 otherwise. The set of crossover points used in a recombination event under punctuated crossover is given by the union of the crossover points specified on each chromosome

compute_mask($\boldsymbol{a}, \boldsymbol{b}$)
  $j := \mathbf{0}$;
  for $i := 1$ **to** $\ell/2$ **do**
  $m_i := j$;
  $m'_i := j$
    if (($a'_i = 1$) or ($b'_i = 1$))
    then $j = 1 - j$;
    fi
  od
  return ($\boldsymbol{m}$);

Note that the symbol and punctuation mark associated with a chromosome position are transmitted together by the punctuated crossover operator. While the idea behind this operator is appealing, empirical tests of punctuated crossover were not conclusive and the operator is not widely used.

In practice, various aspects of these operators are often modified to enhance performance. Consider, for example, the choice of retaining both resultant strings produced by crossover (a common practice) versus discarding one of the offspring. Holland (1975) described an implementation designed to process only one new individual per generation and, consequently, his algorithm discards one of the offspring generated by crossover. Some implementations retain this feature even if they produce more than one new individual per generation. However, empirical studies (Booker 1982) have shown that retaining both offspring can substantially reduce the loss of diversity in the population. Another widespread practice is to restrict the crossover points to those locations where the parent strings have different symbols. This so-called *reduced surrogate* technique (Booker 1987) improves the ability of crossover to produce offspring that are different from their parents.

An implementation technique called *shuffle crossover* was introduced by Eshelman *et al* (1989). The symbols in the parent strings are 'shuffled' by a permutation operator before crossover is invoked. The inverse permutation is applied to the offspring produced by crossover to restore the original symbol ordering. This method can be used to counteract the tendency in n-point crossover ($n \geq 1$) to disrupt sets of symbols that are widely dispersed on the chromosome more than it disrupts symbols which are close together (see the discussion of bias in section C3.3.1.4).

The crossover mechanisms described so far are all consistent with the simplest principle of Mendelian inheritance: the requirement that every gene carried by an offspring is a copy of a gene inherited from one of its parents. Radcliffe (1991) points out that this conservation of genetic material during recombination is not a necessary restriction for artificial recombination operators. From the standpoint of conducting a robust exploration of the opportunities represented by the parent strings, it is reasonable to ask whether a crossover operator can generate all possible offspring having some combination of genes found in the parents. Given a binary string representation, the answer for one-point and n-point crossover is no while the answer for shuffle crossover and uniform crossover is yes. (To see this, simply consider the set of

possible resultant strings for the parents **0** and **1**.) For nonbinary strings, however, the only way to achieve this capability is to allow the offspring to have genes that are not carried by either parent. Radcliffe used this idea as the basis for designing the *random respectful recombination* operator. This operator generates a resultant string by copying the symbols at positions where the parents are identical, then choosing random values to fill the remaining positions. Note that for binary strings, random respectful recombination is equivalent to uniform crossover with $p_x = 0.5$.

C3.3.1.3 Formal analysis

Mathematical characterizations of crossover. Several characterizations of crossover operators have been formulated to facilitate the formal analysis of recombination and genetic algorithms. Geiringer (1944) characterized recombination in terms of the probability that sets of genes are transmitted from parents to offspring during a recombination event. The behavior of a crossover operator is then completely specified by the probability distribution it induces over the set of all possible recombination events. Geiringer's study of these so-called *recombination distributions* includes a thorough analysis of recombination acting on a population of linear chromosomes in the absence of selection.

In more detail, the recombination distribution associated with a crossover operator is defined as follows. Let $S_\ell = \{1, \ldots, \ell\}$ be the set of ℓ numbers designating the loci in strings of length ℓ. The number of alleles allowed at each locus can be any arbitrary integer. For notational convenience we will identify a crossover mask $\boldsymbol{m}$ with the subset $A \subseteq S_\ell$ which indicates the loci corresponding to the bit positions i where $m_i = 1$. The set A is simply another way to designate the recombination event specified by $\boldsymbol{m}$. The complementary subset $A' = S_\ell \setminus A$ designates the recombination event specified by $\mathbf{1} - \boldsymbol{m}$. The recombination distribution $\mathcal{R}$ is given by the probabilities $\mathcal{R}(A)$ for each recombination event. Clearly, under Mendelian segregation, $\mathcal{R}(A) = \mathcal{R}(A')$ since all alleles will be transmitted to one offspring or the other. It is also clear that $\sum_{A \subseteq S_\ell} \mathcal{R}(A) = 1$. We can therefore view recombination distributions as probability distributions over the power set 2^{S_ℓ} (Schnell 1961). The marginal recombination distribution $\mathcal{R}_A$, describing the transmission of the loci in A, is given by the probabilities

$$\mathcal{R}_A(B) = \sum_{C \subseteq A'} \mathcal{R}(B \cup C) \qquad B \subseteq A.$$

$\mathcal{R}_A(B)$ is the marginal probability of the recombination event in which one parent transmits the loci in $B \subseteq A$ and the other parent transmits the loci in $A \setminus B$.

Other mathematical characterizations of crossover operators are useful when the chromosomes happen to be binary strings. If the sum $\boldsymbol{x} \oplus \boldsymbol{y}$ denotes component-wise addition in the group of integers modulo 2 and the product $\boldsymbol{xy}$ denotes bitwise multiplication, then the strings produced by a crossover operator with mask $\boldsymbol{m}$ are given by $\boldsymbol{ma} \oplus (\mathbf{1} - \boldsymbol{m})\boldsymbol{b}$ and $\boldsymbol{mb} \oplus (\mathbf{1} - \boldsymbol{m})\boldsymbol{a}$. Liepins and Vose (1992) use this definition to show that a binary operator is a crossover operator if and only if the operator preserves schemata and commutes with addition and bitwise multiplication. Furthermore, they provide two characterizations of the set of chromosomes that can be generated by an operator given an initial pool of parent strings. Algebraically, this set is given by the mathematical closure of the parent strings under the crossover operator. Geometrically, the set is determined by projections defined in terms of the crossover masks associated with the operator. Liepins and Vose prove that these algebraic and geometric characterizations are equivalent.

The dynamics of recombination. Geiringer used recombination distributions to examine how recombination without selection modifies the proportions of individuals in a population over time. Assume that each individual $x \in \{1, 2, \ldots, k\}^\ell$ is a string of length ℓ in a finite alphabet of k characters. We also assume in the following that $B \subseteq A \subseteq S_\ell$. Let $p^{(t)}(\boldsymbol{x})$ be the frequency of individual $\boldsymbol{x}$ in a population at generation t, and $p_A^{(t)}(\boldsymbol{x})$ denote the marginal frequency of individuals that are identical to $\boldsymbol{x}$ at the loci in A. That is,

$$p_A^{(t)}(\boldsymbol{x}) = \sum_{\boldsymbol{y}} p^{(t)}(\boldsymbol{y}) \qquad \text{for each } \boldsymbol{y} \text{ satisfying } y_i = x_i \ \forall i \in A.$$

Geiringer derives the following important recurrence relations:

$$p^{(t+1)}(\boldsymbol{z}) = \sum_{A,\boldsymbol{x},\boldsymbol{y}} \mathcal{R}(A)\, p^{(t)}(\boldsymbol{x})\, p^{(t)}(\boldsymbol{y}) \qquad \text{where} \begin{cases} A \subseteq S_\ell \text{ is arbitrary} \\ y_i = z_i \ \ \forall i \in A \\ x_i = z_i \ \ \forall i \in A' \end{cases} \tag{C3.3.1}$$

$$p^{(t+1)}(\boldsymbol{z}) = \sum_{A,B,\boldsymbol{x},\boldsymbol{y}} \mathcal{R}_A(B)\, p^{(t)}(\boldsymbol{x})\, p^{(t)}(\boldsymbol{y}) \qquad \text{where} \quad \begin{cases} B \subseteq A \subseteq S_\ell \text{ are arbitrary subsets} \\ x_i \neq z_i,\ y_i = z_i \ \ \forall i \in B \\ x_j = z_j,\ y_j \neq z_j \ \ \forall j \in A \setminus B \\ x_k = y_k = z_k \ \ \forall k \in A' \end{cases} \tag{C3.3.2}$$

$$p^{(t+1)}(\boldsymbol{z}) = \sum_{A \subseteq S_\ell} \mathcal{R}(A)\, p_A^{(t)}(\boldsymbol{z})\, p_{A'}^{(t)}(\boldsymbol{z}). \tag{C3.3.3}$$

These recurrence relations are equivalent, complete characterizations of how recombination changes the proportion of individuals from one generation to the next. Equation (C3.3.1) has the straightforward interpretation that alleles appear in offspring if and only if they appear in the parents and are transmitted by a recombination event. Each term on the right-hand side of (C3.3.1) is the probability of a recombination event between parents having the desired alleles at the loci that are transmitted together. A string $\boldsymbol{z}$ is the result of a recombination event A whenever the alleles of $\boldsymbol{z}$ at loci A come from one parent and the alleles at loci A' come from the other parent. The change in frequency of an individual string is therefore given by the total probability of all these favorable occurrences. Equation (C3.3.2) is derived from (C3.3.1) by collecting terms based on marginal recombination probabilities. Equation (C3.3.3) is derived from (C3.3.1) by collecting terms based on marginal frequencies of individuals.

The last equation is perhaps the most significant, since it leads directly to a theorem characterizing the expected distribution of individuals in the limit.

Theorem (Geiringer's theorem II). If ℓ loci are arbitrarily linked, with the one exception of 'complete linkage', the distribution of transmitted alleles 'converges toward independence'. The limit distribution is given by

$$\lim_{t\to\infty} p^{(t)}(\boldsymbol{z}) = \prod_{i=1}^{\ell} p^{(0)}(z_i)$$

which is the product of the ℓ marginal distributions of alleles from the initial population.

This theorem tells us that, in the limit, random mating and recombination without selection lead to chromosome frequencies corresponding to the simple product of initial allele frequencies. A population in this state is said to be in *linkage equilibrium* or Robbins' equilibrium (Robbins 1918). This result holds for all recombination operators that allow any two loci to be separated by recombination.

Note that Holland (1975) sketched a proof of a similar result for schema frequencies and one-point crossover. Geiringer's theorem applied to schemata gives us a much more general result. Together with the recurrence equations, this work paints a picture of 'search pressure' from recombination acting to reduce departures from linkage equilibrium for all schemata.

Subsequent work has carefully analyzed the dynamics of this convergence to linkage equilibrium (Christiansen 1989). It has been proven, for example, that the convergence rate for any particular schema is given by the probability of the recombination event specified by the schema's defining loci. In this view, an important difference between crossover operators is the rate at which, undisturbed by selective pressures, they drive schemata to their equilibrium proportions. These results from mathematical population genetics have only recently been applied to evolutionary algorithms (Booker 1993, Altenberg 1995).

Disruption analysis. Many formal studies of crossover operators focus specifically on the way recombination disrupts and constructs schemata. Holland's (1975) original analysis of genetic algorithms derived a bound for the disruptive effects of one-point crossover. This bound was based on the probability $\ell(\xi)/(\ell-1)$ that a single crossover point will fall within the defining length $\ell(\xi)$ of a schema ξ. Bridges and Goldberg (1987) subsequently provided an exact expression for the probability of disruption for one-point crossover. Spears and De Jong (1991a) generalized these results to provide exact expressions for the disruptive effects of n-point and uniform crossover.

Recombination distributions provide a convenient framework for analyzing these disruptive effects (Booker 1993). The first step in this analysis is to derive the marginal distributions for one-point, n-point, and uniform crossover. Analyses using recombination distributions can be simplified for binary strings if we represent individual strings using index sets (Christiansen 1989). Each binary string $\boldsymbol{x}$ can be represented uniquely by the subset $A \subseteq S_\ell$ using the convention that A designates the loci where $x_i = 1$ and A' designates the loci where $x_i = 0$. In this notation S_ℓ represents the string $\mathbf{1}$, $\emptyset$ represents the string $\mathbf{0}$, and A' represents the binary complement of the string represented by A. Index sets can greatly

simplify expressions involving individual strings. Consider, for example, the marginal frequency $p_A(\boldsymbol{x})$ of individuals that are identical to $\boldsymbol{x}$ at the loci in A. The index set expression

$$p_A(B) = \sum_{C \subseteq A'} p(B \cup C) \qquad B \subseteq A$$

makes it clear that $p_A(B)$ involves strings having the allele values given by B at the loci designated by A. Note that $p_\emptyset(B) = 1$ and $p_{S_\ell}(B) = p(B)$.

With this notation we can also succinctly relate recombination distributions and schemata. If A designates the defining loci of a schema ξ and $B \subseteq A$ specifies the alleles at those loci, then the frequency of ξ is given by $p_A(B)$ and the marginal distribution $\mathcal{R}_A$ describes the transmission of the defining loci of ξ. In what follows we will assume, without loss of generality, that the elements of the index set A for a schema ξ are in increasing order so that the kth element $A_{(k)}$ is the locus of the kth defining position of ξ. This means, in particular, that the outermost defining loci of ξ are given by the elements $A_{(1)}$ and $A_{(\mathrm{O}(\xi))}$ where $\mathrm{O}(\xi)$ is the order of ξ. It will be convenient to define the following property relating the order of a schema to its defining length $\delta(\xi)$.

Definition. The *kth component of defining length* for schema ξ, $\delta_k(\xi)$, is the distance between the kth and $k+1$st defining loci, $1 \leq k < \mathrm{O}(\xi)$, with the convention that $\delta_0(\xi) \equiv \ell - \delta(\xi)$.

Note that the defining length of a schema is equal to the sum of its defining length components:

$$\delta(\xi) = \sum_{k=1}^{\mathrm{O}(\xi)-1} \delta_k(\xi) = A_{(\mathrm{O}(\xi))} - A_{(1)}.$$

Given these preliminaries, we can proceed to describe the recombination distributions for specific crossover operators.

One-point crossover. Assume exactly one crossover point in a string of length ℓ, chosen between loci i and $i+1$ with probability $1/(\ell-1)$ for $i = 1, 2, \ldots, \ell-1$. The only recombination events with nonzero probability are $S_x = [1, x]$ and $S'_x = [x+1, \ell-1]$ for $x = 1, 2, \ldots, \ell-1$. The probability of each event is

$$\mathcal{R}^1(S_x) = \mathcal{R}^1(S'_x) = \frac{1}{2(\ell-1)}$$

since each parent is equally likely to transmit the indicated loci. The marginal distribution $\mathcal{R}^1_A$ for an arbitrary index set A can be expressed solely in terms of these recombination events. We will refer to these events as the primary recombination events.

Now for any arbitrary event $B \subseteq A$ there are two cases to consider:

(i) $B = \emptyset$. This corresponds to the primary recombination events $S_x,\ x < A_{(1)}$ and $S'_x,\ x \geq A_{(\mathrm{O}(\xi))}$. There are $\ell - 1 - \delta(\xi)$ such events.

(ii) $B \neq \emptyset$. These situations involve the primary events $S_x,\ A_{(1)} \leq x < A_{(\mathrm{O}(\xi))}$. The events B having nonzero probability are given by $B_i = \{A_{(1)}, \ldots, A_{(i)}\},\ 1 \leq i < \mathrm{O}(\xi)$. For each i, there are $\delta_i(\xi)$ corresponding primary events.

The complete marginal distribution is therefore given by

$$\mathcal{R}^1_A(B) = \begin{cases} \dfrac{\ell - 1 - \delta(\xi)}{2(\ell-1)} & \text{if } B = \emptyset \text{ or } B = A \\ \dfrac{\delta_i(\xi)}{2(\ell-1)} & \text{if } B = B_i,\ 1 \leq i < \mathrm{O}(\xi) \\ 0 & \text{otherwise.} \end{cases}$$

Note that if we restrict our attention to disruptive events, we obtain the familiar result

$$1 - (\mathcal{R}^1_A(\emptyset) + \mathcal{R}^1_A(A)) = 1 - 2\left(\frac{\ell - 1 - \delta(\xi)}{2(\ell-1)}\right) = 1 - \left(1 - \frac{\delta(\xi)}{\ell - 1}\right) = \frac{\delta(\xi)}{\ell - 1}.$$

n-point crossover. The generalization to n crossover points in a string of length ℓ uses the standard convention (De Jong 1975) that when the number of crossover points is odd, a final crossover point is defined at position zero. We also assume that all the crossover points are distinct, which corresponds to the way multipoint crossover is often implemented. Given these assumptions, there are $2\binom{\ell}{n}$ nonzero recombination events if n is even or $n = \ell$, and $2\binom{\ell-1}{n}$ such events if n is odd. Since the n points are randomly selected, these events are equally likely to occur.

We derive an expression for the marginal distributions in the same way as we proceeded for one-point crossover. First we identify the relevant recombination events, then we count them up and multiply by the probability of a single event. Identification of the appropriate recombination events begins with the observation (De Jong 1975) that crossover does not disrupt a schema whenever an even number of crossover points (including zero) fall between successive defining positions. We can use this to identify the configurations of crossover points that transmit all the loci in $B \subseteq A$ and none of the loci in $A \setminus B$. Given any two consecutive elements of A, there should be an even number of crossover points between them if they both belong to B or $A \setminus B$. Otherwise there should be an odd number of crossover points between them. This can be formalized as a predicate $\mathcal{X}_A$ that tests these conditions for a marginal distribution $\mathcal{R}_A$

$$\mathcal{X}_A(B, n, i) = \begin{cases} 1 & \text{if } n \text{ is even and } \{A_{(i)}, A_{(i-1)}\} \cap B = \emptyset \text{ or } \{A_{(i)}, A_{(i-1)}\} \\ 1 & \text{if } n \text{ is odd and } \{A_{(i)}, A_{(i-1)}\} \cap B \neq \emptyset \text{ or } \{A_{(i)}, A_{(i-1)}\} \\ & \text{where } 2 \leq i \leq \mathrm{O}(\xi) \\ 0 & \text{otherwise.} \end{cases}$$

The recombination events can be counted by simply enumerating all possible configurations of crossover points and discarding those not associated with the marginal distribution. The following function $\mathcal{N}_A$ computes this count recursively (as suggested by the disruption analysis of Spears and De Jong (1991a)):

$$\mathcal{N}_A(B, n, i) = \begin{cases} \displaystyle\sum_{j=0}^{n} \binom{\delta_{i-1}(\xi)}{j} \mathcal{X}_A(B, j, i) \mathcal{N}_A(B, n-j, i-1) & 2 < i \leq \mathrm{O}(\xi) \\ \binom{\delta_1(\xi)}{n} \mathcal{X}_A(B, n, 2) & i = 2. \end{cases}$$

Putting all the pieces together, we can now give an expression for the complete marginal distribution.

$$\mathcal{R}_A^n(B) = \begin{cases} \dfrac{\displaystyle\sum_{j=0}^{n} \binom{\delta_0(\xi)}{j} \mathcal{N}_A(B, n-j, \mathrm{O}(\xi))}{2\binom{\ell}{n}} & \text{if } n \text{ is even or } n = \ell \\ \dfrac{\displaystyle\sum_{j=0}^{n} \binom{\delta_0(\xi)-1}{j} \mathcal{N}_A(B, n-j, \mathrm{O}(\xi))}{2\binom{\ell-1}{n}} & \text{otherwise.} \end{cases}$$

Uniform crossover. The marginal distribution $\mathcal{R}_A^{u(p)}$ for parametrized uniform crossover with parameter p is easily derived from previous analyses (Spears and De Jong 1991). It is given by

$$\mathcal{R}_A^{u(p)}(B) = p^{|B|}(1-p)^{|A \setminus B|}.$$

Figure C3.3.1 shows how the marginal probability of transmission for second-order schemata—$2\,\mathcal{R}_A^n(A)$ and $2\,\mathcal{R}_A^{u(0.5)}$, $|A| = 2$—varies as a function of defining length. The shape of the curves depends on whether n is odd or even. Since the curves indicate the probability of transmitting schemata, the area above each curve can be interpreted as a measure of potential schema disruption. This interpretation makes it clear that two-point crossover is the best choice for minimizing disruption. Spears and De Jong (1991a) have shown that this property of two-point crossover remains valid for higher-order schemata.

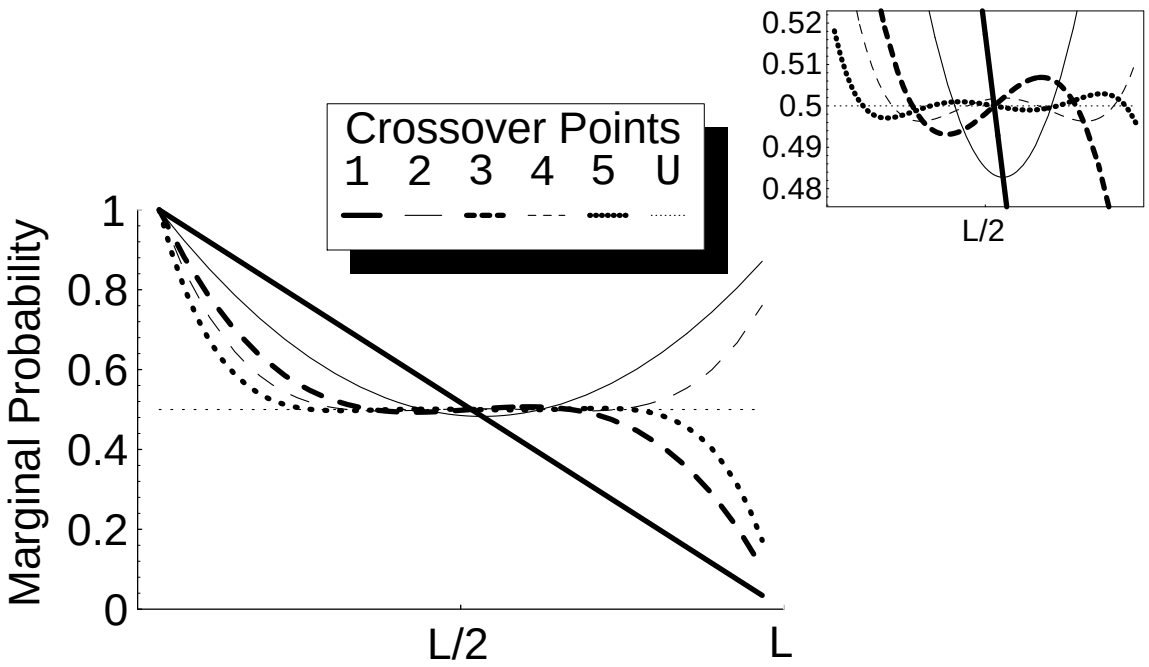


Figure C3.3.1. Transmission probabilities for second-order schemata. The inset shows the behavior of these curves in the vicinity of the point $L/2$.

Note that these curves are not identical to the family of curves for nondisruptive crossovers given by Spears and De Jong. The difference is that Spears and De Jong assume crossover points are selected randomly with replacement. This means that their measure $P_{2,\text{even}}$ is a polynomial function of the defining length having degree n, with n identical solutions to the equation $P_{2,\text{even}} = 1/2$ at the point $\ell/2$. The function $\mathcal{R}_A^n(A)$, on the other hand, has n distinct solutions to the equation $2\,\mathcal{R}_A^n(A) = 1/2$ as shown in the upper right-hand corner of figure C3.3.1. This property stems from our assumption that crossover points are distinct and hence selected without replacement.

Finally, regarding the construction of schema, Holland (1989) has analyzed the expected waiting time to construct a new schema that falls in the intersection of two schemas already established in a population. He gives examples showing that the waiting time for one-point crossover to construct the new schema can be several orders of magnitude shorter than the waiting time for mutation. Thierens and Goldberg (1993) also examine this property of recombination by analyzing so-called *mixing events*—recombination events in which building blocks from the parents are juxtaposed or 'mixed' to produce an offspring having more building blocks than either parent. Using the techniques of dimensional analysis they show that, given only simple selection and uniform crossover, effective mixing requires a population size that grows exponentially with the number and length of the building blocks involved. This indicates that additional mechanisms may be needed to achieve effective mixing in genetic algorithms.

C3.3.1.4 Crossover bias

In order to effectively use any inductive search operator, it is important to understand whatever tendencies the operator may have to prefer one search outcome over another. Any such tendency is called an inductive *bias*. Random search is the only search technique that has no bias. It has long been recognized that an appropriate inductive bias is *necessary* in order for inductive search to proceed efficiently and effectively (Mitchell 1980). Two types of bias have been attributed to crossover operators in genetic search: *distributional bias* and *positional bias* (Eshelman *et al* 1989).

Distributional bias refers to the number of symbols transmitted during a recombination event and

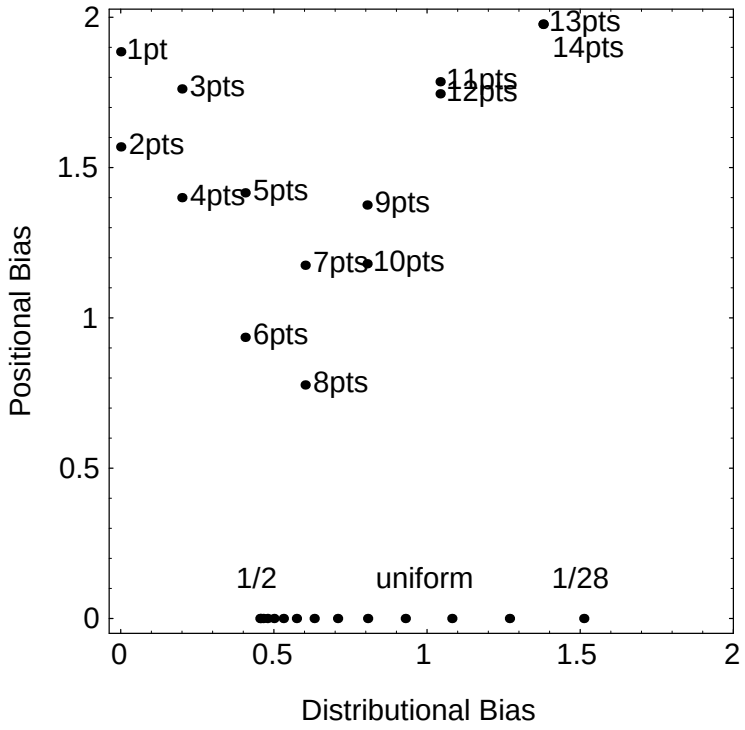


Figure C3.3.2. One view of the crossover bias 'landscape' generated using quantitative measures derived from recombination distributions.

the extent to which some quantities might be more likely to occur than others. This bias is significant because it is correlated with the potential number of schemata from each parent that can be recombined by the crossover operator. An operator has distributional bias if the probability distribution for the number of symbols transmitted from a parent is not uniform. Both one-point and two-point crossover are free of distributional bias. The n-point ($n > 2$) crossover operators have a distributional bias that is well approximated by a binomial distribution with mean $\ell/2$ for large n. Uniform crossover has a strong distributional bias, with the expected number of symbols transmitted given by a binomial distribution with expected value $p_x\ell$. More recently, Eshelman and Schaffer (1995) have emphasized the expected value of the number of symbols transmitted rather than the distribution of those numbers. The bias defined by this criterion, though clearly similar to distributional bias, is referred to as *recombinative bias*.

Positional bias characterizes how much the probability that a set of symbols will be transmitted intact during a recombination event depends on the relative positions of those symbols on the chromosome. This bias is important because it indicates which schemata are likely to be inherited by offspring from their parents. It is also indicative of the extent to which these schemata will appear in new contexts that can help distinguish the genuine instances of co-adaptation from spurious linkage effects. Holland's (1975) analysis of one-point crossover pointed out that the shorter the defining length of a schema, the more likely it is to be transmitted intact during the crossover operation. Consequently, one-point crossover has a strong positional bias. Analyses of n-point crossover (Spears and De Jong 1991a) lead to a similar conclusion for those operators, though the amount of positional bias varies with n (Booker 1993). Uniform crossover has no positional bias, which is one of the primary reasons it is widely used. Note that shuffle crossover was designed to remove the positional bias from one-point and n-point crossover. Eshelman and Schaffer (1995) have revised their view of positional bias, generalizing the notion to something they now call *schema bias*. An operator has no schema bias if schemata of the same order are equally likely to be disrupted regardless of their defining length.

Recombination distributions can be used to derive quantitative measures of crossover bias (Booker 1993). The overall bias 'landscape' for various crossover operators based on these measures is summarized in figure C3.3.2.

C3.3.2 Real-valued vectors

David B Fogel

Abstract

Real-valued vectors can be recombined in a variety of ways to generate new candidate solutions in evolutionary algorithms, and several such methods are presented here.

Recombination acts on two or more elements in a population to generate at least one offspring. When the elements are *real-valued vectors*, recombination can be implemented in a variety of forms. Many of these C1.3 forms derive from efforts within the evolution strategies community because of their long involvement with continuous optimization problems. The simpler versions, however, have been popularized within research in genetic algorithms.

For two parent real-valued vectors $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, each of dimension n, one-point crossover is performed by selecting a random crossover point k and exchanging the elements occurring after point k in $\boldsymbol{x}_1$ with those that occur after point k in $\boldsymbol{x}_2$ (see figures C3.3.3 and C3.3.4). This operator can be extended to a two-point crossover in which two crossover points k_1 and k_2 are selected at random and the segment in between these points is exchanged between parents (see figure C3.3.5). Extensions to greater multiple-point crossover operators follow naturally.

Parents

$$\boldsymbol{x}_1 = x_{1,1}x_{1,2}\ldots x_{1,k}x_{1,k+1}\ldots x_{1,d}$$
$$\boldsymbol{x}_2 = x_{2,1}x_{2,2}\ldots x_{2,k}x_{2,k+1}\ldots x_{2,d}$$

Offspring

$$\boldsymbol{x}'_1 = x_{1,1}x_{1,2}\ldots x_{1,k}x_{2,k+1}\ldots x_{2,d}$$
$$\boldsymbol{x}'_2 = x_{2,1}x_{2,2}\ldots x_{2,k}x_{1,k+1}\ldots x_{1,d}$$

Figure C3.3.3. For one-point crossover, two parents are chosen and a crossover point k is selected, typically uniformly across the components. Two offspring are created by interchanging the segments of the parents that occur from the crossover point to the ends of the string.

The one-point and two-point operators attempt to recombine vector segments. Alternatively, individual elements can be recombined without regard to longer segments in which they reside by using a uniform recombination operator. Given two parents $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, one or more offspring are created by randomly selecting each next element from either parent (see figure C3.3.6). Typically, each parent has an equal chance of contributing the next element. This procedure was offered early on by Reed *et al* (1967) and was reintroduced within the genetic algorithm community by Syswerda (1989). A similar procedure is also used within evolution strategies and termed 'discrete recombination' (see below, and also see the *uniform scan* operator of Eiben *et al* (1994), which is applied to multiple parents).

In contrast to the crossover type recombination operators that exchange information between parents, intermediate recombination operators attempt to average or blend components across multiple parents. A canonical version acts on two parents $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, and creates an offspring $\boldsymbol{x}'$ as the weighted average:

$$x'_i = \alpha x_{1i} + (1-\alpha)x_{2i}$$

where α is a number in $[0, 1]$ and $i = 1, \ldots, n$ (figure C3.3.7). If $\alpha = 0.5$, then the operation is a simple average at each component. Note that this operator can be extended to act on more than two parents (i.e. a multirecombination) by the operation

$$x'_i = \alpha_1 x_{1i} + \alpha_2 x_{2i} + \ldots + \alpha_k x_{ki}$$

subject to

$$\sum \alpha_i = 1 \qquad i = 1, \ldots, k$$

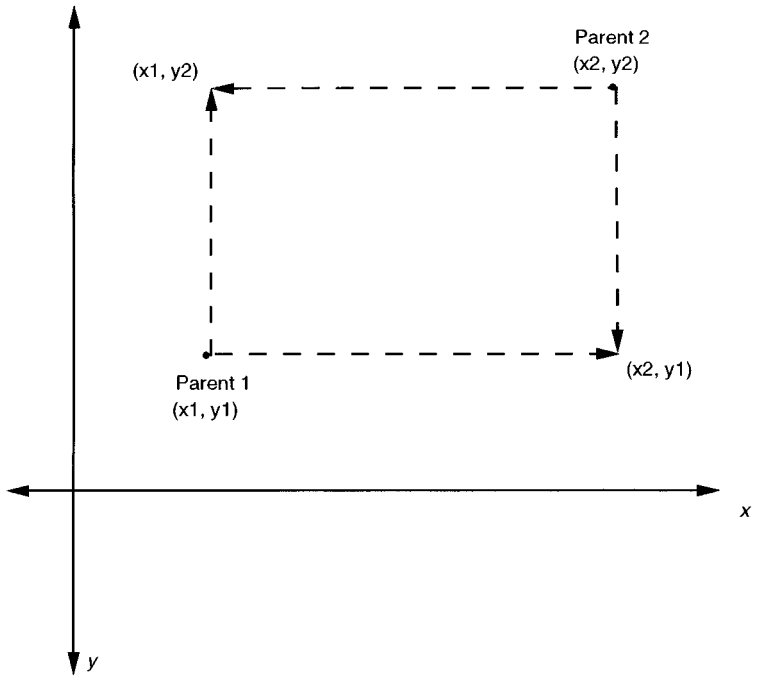


Figure C3.3.4. A two-dimensional illustration of the potential offspring under a one-point crossover operator applied to real-valued parents.

Parents

$$\boldsymbol{x}_1 = x_{1,1}x_{1,2}\ldots x_{1,k_1}x_{1,k_1+1}\ldots x_{1,k_2}x_{1,k_2+1}\ldots x_{1,d}$$
$$\boldsymbol{x}_2 = x_{2,1}x_{2,2}\ldots x_{2,k_1}x_{2,k_1+1}\ldots x_{2,k_2}x_{2,k_2+1}\ldots x_{2,d}$$

Offspring

$$\boldsymbol{x}'_1 = x_{1,1}x_{1,2}\ldots x_{2,k_1}x_{2,k_1+1}\ldots x_{2,k_2}x_{1,k_2+1}\ldots x_{2,d}$$
$$\boldsymbol{x}'_2 = x_{2,1}x_{2,2}\ldots x_{1,k_1}x_{1,k_1+1}\ldots x_{1,k_2}x_{2,k_2+1}\ldots x_{2,d}$$

Figure C3.3.5. For two-point crossover, two parents are chosen and two crossover points, k_1 and k_2, are selected, typically uniformly across the components. Two offspring are created by interchanging the segments defined by the points k_1 and k_2.

where there are k individuals involved in the multirecombination. This general procedure is also known as *arithmetic crossover* (Michalewicz 1996, p 112) and has been described in various other terms in the literature.

In a more generalized manner, recombination operators can follow the following forms (Bäck *et al* 1993, Fogel 1995 pp 146–7):

$$x'_i = \begin{cases} x_{\mathrm{S},i} & \text{(C3.3.1)} \\ x_{\mathrm{S},i} \quad \text{or} \quad x_{\mathrm{T},i} & \text{(C3.3.2)} \\ x_{\mathrm{S},i} + u(x_{\mathrm{T},i} - x_{\mathrm{S},i}) & \text{(C3.3.3)} \\ x_{\mathrm{S}_j,i} \quad \text{or} \quad x_{\mathrm{T}_j,i} & \text{(C3.3.4)} \\ x_{\mathrm{S}_j,i} + u_i(x_{\mathrm{T}_j,i} - x_{\mathrm{S}_j,i}) & \text{(C3.3.5)} \end{cases}$$

where S and T denote two arbitrary parents, u is a uniform random variable over [0, 1], and i and j index the components of a vector and the vector itself, respectively. The versions are no recombination (C3.3.4), discrete recombination (or uniform crossover) (C3.3.5), intermediate recombination (C3.3.6), and (C3.3.7)

Parents

$$\boldsymbol{x}_1 = x_{1,1}x_{1,2}\ldots x_{1,d}$$
$$\boldsymbol{x}_2 = x_{2,1}x_{2,2}\ldots x_{2,d}$$

Offspring

$$\boldsymbol{x}'_1 = x_{1,1}x_{2,2}\ldots x_{1,d}$$
$$\boldsymbol{x}'_2 = x_{1,1}x_{1,2}\ldots x_{2,d}$$

Figure C3.3.6. For uniform crossover, each element of an offspring (here two offspring are depicted) is selected from either parent. The example shows that the first element in both offspring were selected from the first parent. In some applications such duplication is not allowed. Typically each parent has an equal chance of contributing each element to an offspring.

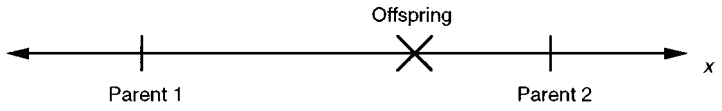


Figure C3.3.7. A geometrical interpretation of intermediate recombination applied to two parents in a single dimension.

and (C3.3.8) are the global versions of (C3.3.5) and (C3.3.6), respectively, extended to include more than two parents (up to as many as the entire population size).

There are several other variations of crossover operators that have been applied to real-valued vectors.

(i) The *heuristic crossover* of Wright (1994) takes the form

$$\boldsymbol{x}' = u(\boldsymbol{x}_2 - \boldsymbol{x}_1) + \boldsymbol{x}_2$$

where u is a uniform random variable over [0, 1] and $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ are the two parent vectors subject to the condition that $\boldsymbol{x}_2$ is not worse than $\boldsymbol{x}_1$. Michalewicz (1996, p 129) noted that this operator uses values of the objective function to determine a direction to search.

(ii) The *simplex crossover* of Renders and Bersini (1994) selects $k > 2$ parents (say the set J), determines the best and worst individuals within the selected group (say $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, respectively), computes the centroid of the group without $\boldsymbol{x}_2$ (say $\boldsymbol{c}$) and computes the reflected vector $\boldsymbol{x}'$ (the offspring) obtained from the vector $\boldsymbol{x}_2$ as

$$\boldsymbol{x}' = \boldsymbol{c} + (\boldsymbol{c} - \boldsymbol{x}_2).$$

(iii) The *geometrical crossover* of Michalewicz *et al* (1996) takes two parents $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ and produces a single offspring $\boldsymbol{x}'$ as

$$\boldsymbol{x}' = [(x_{11}x_{21})^{0.5}, \ldots, (x_{1n}x_{2n})^{0.5}].$$

This operator can be generalized to a multiparent version:

$$\boldsymbol{x}' = [(x_{11}^{\alpha^1}x_{21}^{\alpha^2}\ldots x_{k1}^{\alpha^k}), \ldots, (x_{1n}^{\alpha^1}x_{2n}^{\alpha^2}\ldots x_{kn}^{\alpha^k})].$$

(iv) The *fitness-based scan* of Eiben *et al* (1994) takes multiple parents and generates an offspring where each component is selected from one of the parents with a probability corresponding to the parent's relative fitness. If a parent has fitness $f(i)$, then the likelihood of selecting each individual component from that parent is $f(i)/\sum f(j)$, where $j = 1, \ldots, k$ and there are k parents involved in the operator.

(v) The diagonal multiparent crossover of Eiben *et al* (1994) operates much like n-point crossover, except that in creating k offspring from k parents, $c \geq 1$ crossover points are chosen and the first offspring is constructed to contain the first segment from parent 1, the second segment from parent 2, and so forth. Subsequent offspring are similarly constructed from a rotation of segments from the parents.

C3.3.3 Permutations

Darrell Whitley

Abstract

Several different recombination operators have been designed for application to problems represented as permutations. This includes operators for the traveling salesman problem and vehicle routing, as well as various scheduling applications.

C3.3.3.1 Introduction

An obvious attribute of *permutation problems* is that simple crossover operators fail to generate offspring that are permutations. Consider the following example of simple one-point crossover, when one parent is denoted with capital letters and the other with lower-case letters: C1.4

```
String 1:  A B C  D E F G H I
              \/
              /\
String 2:  h d a  e i c f b g

Offspring 1:  A B C e i c f b g
Offspring 2:  h d a D E F G H I.
```

Neither of the two offspring represents a legal permutation. Offspring 1 duplicates elements *B* and *C* while omitting elements *H* and *D*. Offspring 2 has just the opposite problem: it duplicates *H* and *D* while omitting *B* and *C*.

Davis (1985) and Goldberg and Lingle (1985) defined some of the first operators for permutation problems. One variant of Davis' order crossover operator can be described as follows.

Davis' order crossover. Pick two permutations for recombination. Denote the first parent as the *cut string* and the other the *filler string.* Select two *crossover* points. Copy the sublist of permutation elements between the crossover points from the cut string directly to the offspring, placing them in the same absolute position. This will be referred to as the crossover section. Next, *starting at the second crossover point*, find the next element in the filler string that does not appear in the offspring. Starting at the second crossover point, place the element from the filler string into the next available slot in the offspring. Continue moving the next unused element from the filler string to the offspring. When the end of the filler string (or the offspring) is reached, wrap around to the beginning of the string. When done in this way, Davis' order crossover has the property that Radcliffe (1994) describes as pure recombination: when two identical parents are recombined the offspring will also be identical with the parents. If one does not start copying elements from the filler string starting at the second crossover point, the recombination may not be pure.

The following is an example of Davis' order crossover, where dots represent the crossover points. The underscore symbol in the crossover section corresponds to empty slots in the offspring.

```
                    Parent 1:  A B . C D E F . G H I
          crossover-section:   _ _   C D E F   _ _ _

                    Parent 2:  h d . a e i c . f b g
available elements in order:   b g h a i

                   Offspring:  a i C D E F b g h.
```

Note that the elements in the crossover section preserve relative order, absolute position, and adjacency from parent 1. The elements that are copied from the filler string preserve only the relative order information from the second parent.

Partially mapped crossover (PMX). Goldberg and Lingle (1985) introduced the partially mapped crossover operator (PMX). PMX shares the following attributes with Davis's order crossover. One parent

string is designed as parent 1, the other as parent 2. Two crossover sites are selected and all of the elements in parent 1 between the crossover sites are directly copied to the offspring. This means that PMX also defines a crossover section in the same manner as order crossover.

```
          Parent 1:  A B . C D E . F G
crossover-section:   _ _   C D E   _ _

          Parent 2:  c f . e b a . d g.
```

The difference between the two operators is in how PMX copies elements from parent 2 into the open slots in the offspring after a crossover section has been defined. Denote the parents as P1 and P2 and the offspring as OS; let $P1_i$ denote the ith element of permutation P1. The following description of selecting elements from P2 to place in the offspring is based on the article by Whitley and Yoo (1995).

For those elements between the crossover points in parent 2, if element $P2_i$ has already been copied to the offspring, take no action. In the example given here, element *e* in parent 2 requires no processing. We will consider the rest of the elements by considering the positions in which they appear in the crossover section. If the next element at position $P2_i$ in parent 2 has not already been copied to the offspring, then find $P1_i = P2_j$; if position j has not been filled in the offspring then assign $OS_j = P2_i$. In the example given here, the next element in the crossover section of parent 2 is *b* which is in the same position as *D* in parent 1. Element *D* is located in parent 2 with index 6 and the offspring at OS_6 has not been filled. Copy *b* to the offspring in the corresponding position. This yields

```
Offspring:    _ _ C D E b _.
```

A problem occurs when we try to place element *A* in the offspring. Element *A* in parent 2 maps to element *E* in parent 1; *E* falls in position 3 in parent 2, but position 3 has already been filled in the offspring. The position in the offspring is filled by *C*, so we now find element *C* in parent 2. The position is unoccupied in the offspring, so element *A* is placed in the offspring at the position occupied by *C* in parent 2. This yields

```
Offspring:    a _ C D E b _.
```

All of the elements in parent 1 and parent 2 that fall within the crossover section have now been placed in the offspring. The remaining elements can be placed by directly copying their positions from parent 2. This yields

```
Offspring:    a f C D E b g.
```

C3.3.3.2 Order and position crossover

Syswerda's (1991) order crossover-2 and position crossover are different from either PMX or Davis' order crossover in that there is no contiguous block which is directly passed to the offspring. Instead several elements are randomly selected by absolute position.

Order crossover-2. This operator starts by selecting K random positions in parent 2, where the parents are of length L. The corresponding elements from parent 2 are then located in parent 1 and reordered so that they appear in the same relative order as they appear in parent 2. Elements in parent 1 that do not correspond to selected elements in parent 2 are passed directly to the offspring. For example,

```
          Parent 1:  A B C D E F G
          Parent 2:  C F E B A D G
Selected Elements:     *   * *.
```

The selected elements in parent 2 are F, B, and A. Thus, the relevant elements are reordered in parent 1.

```
Reorder  A B _ _ _ F _  from parent 1 which yields  f b _ _ _ a _.
```

All other elements are copied directly from parent 1.

```
(f b _ _ _ a _)  combined with (_ _ C D E _ G) yields  f b C D E a G.
```

Position crossover. Syswerda defines a second operator called *position* crossover. Using the same example that was used to illustrate Syswerda's order crossover-2, first pick $L - K$ elements from parent 1 which are to be directly copied to the offspring. These elements are copied by position. This yields

```
_ _ C D E _ G.
```

Next scan parent 2 from left to right and place place each element which does not yet appear in the offspring in the next available position. This yields the following progression:

```
# # C D E # G  =>  f # C D E # G
               =>  f b C D E # G
               =>  f b C D E a G.
```

Obviously, in this case the two operators generate exactly the same offspring. Jim Van Zant first pointed out the similarity of these two operators in the electronic newsgroup *The Genetic Algorithm Digest.* Whitley and Yoo (1995) show the two operators to be identical using the following argument.

Assume there is one way to produce a target string S by recombining two parents. Given a pair of strings which can be recombined to produce string S, the probability of selecting the K key positions using order crossover-2 required to generate a specific string S is $\binom{L}{K}^{-1}$, while for position crossover the probability of picking the $L - K$ key elements that will produce exactly the same effect is $\binom{L}{L-K}^{-1}$. Since $\binom{L}{K} = \binom{L}{L-K}$ the probabilities are identical.

Now assume there are R unique ways to recombine two strings to generate a target string S. The probabilities for each unique recombination event are equal as shown by the argument in the preceding paragraph. Thus the sum of the probabilities for the various ways of ways of generating S are equivalent for order crossover 2 and position crossover. Since the probabilities of generating any string S are identical, the operators are identical in expectation.

This also means that in practice there is no difference between using order crossover-2 and position crossover as long as the parameters of the operators are adjusted to reflect their complementary nature. If position crossover is used so that $\mathcal{X}$% of the positions are initially copied to the offspring, then order crossover is identical if $(100 - \mathcal{X})$% positions are selected as relative order positions.

C3.3.3.3 Uniform crossover

Davis' uniform crossover (Davis 1991, p 80) is identical to position crossover and order crossover-2, except that two offspring are generated. A bitstring is used to denote the selection of positions. Offspring 1 copies the elements directly from parent 1 in those positions in the bitstring marked by a '1' bit. Offspring 2 copies the elements from parent 2 in those positions marked by '0' bits. Both offspring then copy the remaining elements from the other parent in relative order.

C3.3.3.4 Edge recombination

Edge recombination was introduced as a specialized operator for the *traveling salesman problem* (TSP). The motivation behind this operator is that it should preserve the adjacency between permutation elements, since the cost of a tour in a TSP is directly related to the set of adjacency relationships (i.e. the distances between cities) that exists between permutation elements. The original edge recombination operator has gone through three revisions and enhancements over the years. First, the basic idea behind edge recombination is introduced. G9.5

Since adjacency information directly translates into cost, the adjacency information from two parent strings is extracted and stored in an adjacency list called the edge table. The edge table really just combines the two tours into a single graph. Recombination occurs by building an offspring using the adjacency information stored in the edge table; in other words, it tries to find a new Hamiltonian circuit in the graph created by merging the two parent strings. Finding a Hamiltonian circuit in an arbitrary graph is itself a nondeterministic-polynomial-time (NP) complete problem and edge recombination must sometimes add edges not contained in the edge table in order to generate a legal tour. The various enhancements to edge recombination attempt to reduce the number of 'foreign edges' (edges not found in the edge table) that must be introduced into an offspring during recombination in order to maintain a feasible tour.

In the original edge recombination operator, no information was maintained about common edges that were shared by both parents. As a result the operator sometimes failed to place an edge in the offspring that appeared in both parents, resulting in a kind of 'mutation by omission' (Whitley *et al* 1991). To

solve this problem, information about shared edges was added to the edge table. Edges shared by the two parents are marked with a + symbol. The algorithm can be described as follows.

Consider the following tours as parents to be recombined:

parent 1: g d m h b j f i a k e c parent 2: c e k a g b h i j f m d.

An edge list is constructed for each city in the tour. The edge list for some city *a* is composed of all of the cities in the two parents that are adjacent to city *a*. If some city is adjacent to *a* in both parents, this entry is flagged (using a plus sign). Figure C3.3.8 shows the edge table which is the collective set of edge lists for all cities.

city	edge list
a	+k, g, i
b	+h, g, j
c	+e, d, g
d	+m, g, c
e	+k, -c
f	+j, m, i

city	edge list
g	a, b, c, d
h	+b, i, m
i	h, j, a, f
j	+f, i, b
k	+e, +a
m	+d, f, h

Figure C3.3.8. An edge table.

The algorithm for edge recombination is as follows.

(i) Pick a random city as the initial *current city*. Remove all references to this city from the edge table.

(ii) Look at the adjacency list of the current city. If there is a common edge (flagged by +), go to that city next. (Unless the initial city is the current city, there can be only one common edge; if two common edges existed, one was used to reach the current city.) Otherwise from the cities on the current adjacency list pick the next city to be the one whose own adjacency list is shortest. Ties are broken randomly. Once a city is visited, references to the city are removed from the adjacency list of other cities and it is no longer reachable from other cities.

(iii) Repeat step 2 until the tour is complete or a city has been reached that has no entries in its adjacency list. If not all cities have been visited, randomly pick a new city to start a new partial tour.

Using the edge table in figure C3.3.8, city *a* is randomly chosen as the first city in the tour. City *k* is chosen as the second city in the tour since the edge (a, k) occurs in both parent tours. City *e* is chosen from the edge list of city *k* as the next city in the tour since this is the only city remaining in *k*'s edge list. This procedure is repeated until the partial tour contains the sequence [a k e c].

At this point there is no deterministic choice for the fifth city in the tour. City *c* has edges to cities *d* and *g*, which both have two unused edges remaining. Therefore city *d* is randomly chosen to continue the tour. The normal deterministic construction of the tour then continues until position 7. At position 7 another random choice is made between cities *f* and *h*. City *h* is selected and the normal deterministic construction continues until we arrive at the following partial tour: [a k e c d m h b g].

In this situation, a failure occurs since there are no edges remaining in the edge list for city *g*. When a potential failure occurs during *edge-3* recombination, we attempt to continue construction at a previously unexplored terminal point in the tour.

A *terminal* is a city which occurs at either end of a partial tour, where all edges in the partial tour are inherited from the parents. The terminal is said to be *live* if that city still has entries in its edge list; otherwise it is said to be a *dead* terminal. Because city *a* was randomly chosen to start the tour in the previous example, it serves as a new terminal in the event of a failure. Conceptually this is the same as inverting the partial tour to build from the other end.

When a failure occurs, there is at most one *live* terminal in reserve at the opposite end of the current partial tour. In fact, it is not guaranteed to be live, since the construction of the partial tour could isolate this terminal city. Once both terminals of the current partial tour are found to be dead, a new partial tour must be initiated. Note that no local information is employed.

We now continue construction of the partial tour [a k e c d m h b g]. The tour segment is reversed (i.e. [g b h m d c e k a]). Then city *i* is added to the tour after city *a*. The tour is then constructed in the normal fashion. In this case, there are no further failures. The final offspring tour is [g b h m d c e k a i f j]. The offspring produced has a single foreign edge (i.e. [j–g].)

When a failure occurs at both ends of the subtour, edge-3 recombination starts a new partial tour. However, there is one other possibility, which has been described as part of the edge-4 operator (Dzubera and Whitley 1994) but which has not been widely tested.

Assume that the first partial tour has been constructed such that both ends of the construction lack a *live terminal* by which to continue. Since only one partial tour has been constructed and since initially every city has at least two edges in the edge table, there must be edges internal to the current partial tour that represent possible edges to the *terminal cities* of the partial tour. The edge-4 operator attempts to exploit this fact by inverting part of the partial tour so that a terminal city is reconnected to an edge which is both internal to the partial tour and which appeared in the original edge list of the terminal city. This will cause a previously visited city in the partial tour to move to a terminal position. If this newly created terminal has cities remaining in its (old) edge list, the offspring construction can continue. If it does not, one can look for other internal edges that will allow an inversion. Details on the edge-4 recombination operator are given by Dzubera and Whitley (1994).

If one is using just a recombination operator and a mutation operator, then edge recombination works very well as an operator for the TSP, at least compared to other recombination operators, but if one is hybridizing such that tours are being produced by recombination, then improved using 2-opt, then both the empirical and the theoretical evidence suggests that Mühlenbein's MPX operator may be more effective (Dzubera and Whitley 1994).

C3.3.3.5 Maximal preservative crossover

Mühlenbein (1991, p 331) offers the following pseudocode for the maximal preservative crossover (MPX) operator. (Numbering of the pseudocode has been added for clarity.)

PROC crossover(receiver, donor, offspring)

(i) Choose position $0 <= i <$ nodes and length $b_{low} <= k <= b_{up}$ randomly.
(ii) Extract the string of edges from position i to position $j = (i + k)$ MOD nodes from the mate (donor). This is the crossover string.
(iii) Copy the crossover string to the offspring.
(iv) Add successively further edges until the offspring represents a valid tour. This is done in the following way.
  (a) IF an edge from the receiver parent starting at the last city in the offspring is possible (does not violate a valid tour)
  (b) THEN add this edge from the receiver
  (c) ELSE IF an edge from the donor starting at the last city in the offspring is possible
  (d) THEN add this edge from the donor
  (e) ELSE add that city from the receiver which comes next in the string; this adds a new edge, which we will mark as an implicit mutation.

The following example illustrates the MPX operator.

```
          receiver:    G D M H B J F I A K E C
             donor:    c e k a g b h i j f m d
   initial segment:    _ _ k a g _ _ _ _ _ _ _.
```

Note that G is connected to D in the receiver, and that element D through element I can be taken from the receiver without duplicating any of the elements already in the offspring. This produces the partial tour

```
_ _ k a g D M H B J F I.
```

At this point, there is no edge in either parent that is connected to I and has that not been used. Here MPX skips cities in the receiver until it finds one which has not been used. In this case, it reaches E. This causes E and C to be added to the tour to yield

```
E C k a g D M H B J F I.
```

Note that MPX does not transmit adjacency information from parents to offspring as effectively as the various edge recombination operators, since it uses less lookahead to avoid a break in the tour construction. At the same time, when it must introduce a new edge that does not appear in either parent, it skips to a nearby city in the tour rather than picking a random edge. Assuming that the tour is partially optimized (for example, if the tour has been improved via 2-opt) then a city nearby in the tour should also be a city nearby in Euclidean space. This, coupled with the fact that an initial segment is copied from one of the parents, appears to give MPX an advantage when when combined with an operator such as 2-opt. Gorges-Schleuter (1989) implemented a variant of MPX that has some notable features that are somewhat like Davis's order crossover operator. A full description of Gorges-Schleuter's MPX is given by Dzubera and Whitley (1994).

C3.3.3.6 Cycle crossover

The operators discussed so far are aimed at preserving adjacency information (such as edge recombination) or relative order information (such as Davis' uniform order-based crossover). Operators may also emphasize position. Cycle crossover partitions two parents into a set of cycles: a cycle is a subset of elements which is located on a corresponding subset of positions on both the two parent strings. Consider the following example from Oliver *et al* (1987) where the permutation elements correspond to the alphabetic characters with numbers to indicate position:

```
Parent 1        h k c e f d b l a  i  g  j
Parent 2        a b c d e f g h i  j  k  l
Positions       1 2 3 4 5 6 7 8 9 10 11 12.
```

To find a cycle, pick a position from either parent. Starting with position 1, elements (h, a) belong to cycle 1. The elements (h, a) also appear in positions 8 and 9. Thus the cycle is expanded to include positions (1, 8, 9) and the new elements i and l are added to the corresponding subset. Elements i and l appear in positions 10 and 12, which also causes j to be added to the subset of elements in the cycle. Note that adding j adds no new elements, so the cycle terminates. Cycle 1 includes elements (h, a, i, l, j) in positions (1, 8, 9, 10, 12).

Note that element (c) in position 3 forms a unary cycle of one element. Aside from the unary cycle at element c (denoted U), Oliver *et al* note that there are three cycles between this set of parents:

```
Parent 1        h k c e f d b l a i g j
Parent 2        a b c d e f g h i j k l
Cycle Label     1 2 U 3 3 3 2 1 1 1 2 1.
```

Recombination can occur by picking some cycles from one parent and the remaining cycles from the alternate parent. Note that all elements in the offspring occupy the same positions as in one of the two parents. However, few applications seem to be position sensitive and cycle crossover is less effective at preserving adjacency information (as in the TSP) or relative order information (as in resource scheduling) compared to other operators.

C3.3.3.7 Merge crossover

Blanton and Wainwright (1993) construct permutation recombination operators for multiple vehicle routing with time and capacity constraints. The following example of the merge crossover operator MX1 uses a global precedence vector. Given any two elements in the permutation, the global precedence vector indicates which element has higher priority for processing. Elements which appear earlier in the vector have higher precedence. In vehicle routing each customer has a time window in which they must be served, which can be translated into a global precedence vector: for example, customer X should be served before customer Y because the time window for X closes before the time window for Y. The following example illustrates the operator:

```
  Parent 1: C F G B A H D I E J
  Parent 2: E B G J D I C A F H
Precedence: A B C D E F G H I J.
```

A single offspring is constructed. In this case, starting at position 1, we compare C and E from the two parents; since C has higher precedence, it is placed in the offspring. Because C has already been allocated a position in the offspring, the C which appears later in parent 2 is exchanged with the E in the initial position of parent 2. This yields

```
  Parent 1: C F G B A H  D  I E J
  Parent 2: C B G J D I <E> A F H
Precedence: A B C D E F  G  H I J
```

where the moved E element is bracketed: ¡E¿. Going to position 2, B has higher precedence than F, so B is kept in position 2. Also, elements F and B are exchanged in parent 1, which yields

```
 Parent 1: C B G <F> A H  D  I E J
 Parent 2: C B G  J  D I <E> A F H
Precedence: A B C  D  E F  G  H I J.
```

Note that one need not actually build a separate offspring, since both parents are in effect transformed into copies of the same offspring. The resulting offspring in the above example is

```
Offspring: C B G F A H D E I J.
```

The MX-2 operator is similar, except that when an element is added to the offspring it is deleted from both parents instead of being swapped. Thus, the process works as follows:

```
  Parent 1: C F G B A H D I E J
  Parent 2: E B G J D I C A F H
Precedence: A B C D E F G H I J.
```

C is added to the offspring and deleted from both parents

```
  Parent 1: _ F G B A H D I E J
  Parent 2: E B G J D I _ A F H
 Offspring: C.
```

Instead of now moving to the second element of each permutation, the first remaining elements in the parents are compared: in this case, E and F are the first elements and E is chosen and deleted. The parents are now represented as follows:

```
  Parent 1: _ F G B A H D I _ J
  Parent 2: _ B G J D I _ A F H
 Offspring: C E.
```

Element B is chosen to fill position 3 in the offspring, and the construction continues to produce the offspring

```
Offspring: C E B F G A H D I J.
```

Note that, over time, this class of operator will produce offspring that are closer to the precedence vector—even if no selection is applied.

C3.3.3.8 Some other operators

Other interesting operators have been introduced over the years for permutation problems. Fox and McMahon (1991) introduced an intersection operator that extracts features common to both parents. Eshelman (1991) used a similar strategy to build a recombination operator that extracts all common subtours for the TSP, and assigns all other elements using local search (2-opt) over an otherwise random assignment. Fox and McMahon also constructed a union operator. In this case, each permutation is converted into a binary matrix representation and the offspring is the logical-or of the matrices representing the parents.

Radcliffe and Surry (1995) have also introduced new operators for the TSP, largely by looking at different representations and then defining appropriate operators with respect to the representations. These representations include the permutation representation, the undirected edge representation, the directed edge representation, and the corner representation.

C3.3.4 Finite-state machines

David B Fogel

Abstract

Finite-state machines can be recombined within evolutionary algorithms, and alternative methods for accomplishing such recombination are given here.

Recombination can be applied to logical structures such as finite-state machines. There have been a variety of proposals to accomplish this in the literature. Recall that a *finite-state machine* is a 5-tuple: C1.5

$$M = (Q, T, P, s, o)$$

where Q is a finite set, the set of states, T is a finite set, the set of input symbols, P is a finite set, the set of output symbols, $s : Q \times T \rightarrow Q$, the next state function, and $o : Q \times T \rightarrow P$, the next output function. Perhaps the earliest proposal to recombine finite-state machines in simulated evolution can be found in the work of Fogel (1964) and Fogel *et al* (1966, pp 21–3). The following extended quotation (Fogel *et al* 1966, p 21) may be insightful:

> The recombination of individuals of opposite sex appears to benefit natural evolution. By analogy, why not retain worthwhile traits that have survived separate evolution by combining the best surviving machines through some genetic rule; mutating the product to yield offspring? Note that there is no need to restrict this mating to the best two surviving 'individuals'. In fact, the most obvious genetic rule, majority logic, only becomes meaningful with the combination of more than two machines.

Fogel *et al* (1966) suggested drawing a single state diagram which expresses the majority logic of an array of finite-state machines. Each state of the majority logic machine is the composite of a state from each of the original machines. Thus the majority machine may have a number of states as great as the product of the number of states in the original machines. Each transition of the majority machine is described by that input symbol which caused the respective transition in the original machines, and by that output symbol which results from the majority element logic being applied to the output symbols from each of the original machines (figure C3.3.9). If there are only two parents to recombine in this manner, the majority logic machine reduces to the better of the two parents.

Zhou and Grefenstette (1986) used recombination on finite-state automata applied to binary sequence induction problems. The finite-state automata were defined in terms of a 5-tuple:

$$(Q, S, \delta, q_0, F)$$

where Q is a finite set of states, S is a finite input alphabet, $q_0 \in Q$ is the initial state, δ is the transition function mapping the Cartesian product of Q and S into Q, and F is the set of final states, a subset of Q. The chosen representation was

$$(X_1, Y_1, F_1), (X_2, Y_2, F_2), \ldots, (X_8, Y_8, F_8)$$

where each (X_i, Y_i, F_i) represented the state i, X_i and Y_i corresponded to the destination state of the zero and one arrows from state i, respectively, and F_i was a three-bit code where the first two bits were used to indicate whether or not there existed an arrow from state i, and the third bit showed whether the state i was a final state. The maximum number of states was set to eight. The details of how recombination was implemented on this representation are not obvious from the article by Zhou and Grefenstette (1986) but it is reasonable to infer that a simple one-point crossover operator was applied.

Fogel and Fogel (1986) used recombination in a similar manner on finite-state machines by exchanging single states between machines (i.e. output symbol and next-state transitions for each input symbol for a particular state). Birgmeier (1996) also used a similar method implemented as uniform crossover between two machines by state. One offspring was produced from two parents by choosing each row in the transition table from either parent (with specific procedures for handling parents with differing numbers of states). Birgmeier (1996) also offered a new *joining* operator where the offspring's size is the sum of the two parents' number of states. Both the output and transition matrices from the two parents are juxtaposed in the offspring and some of the entries are randomly reset to point to a state in the other half, thus joining the new machines into one.

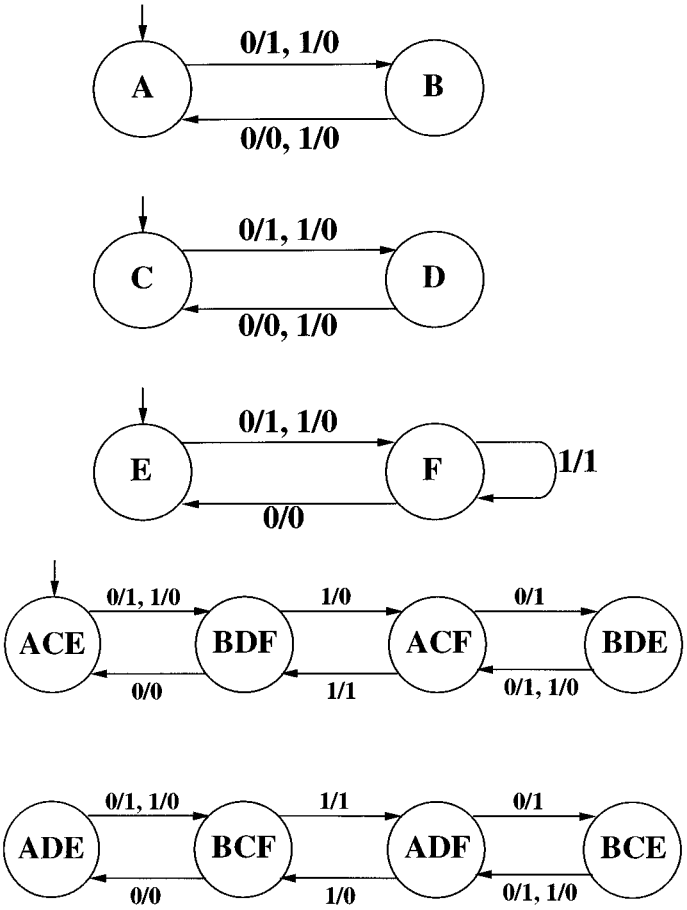


Figure C3.3.9. Three parent machines (top) are joined by a majority logic operator to form another machine (bottom). The initial state of each machine is indicated by a short arrow pointing to that state. Each state in the majority logic machine is a combination of the states of the three parent machines with the output symbol being chosen as the majority decision of two of the three parent machines. For example, the state BDF in the majority logic machine is determined by examining the states B, D, and F in each of the individual machines. For an input symbol of 0, all three states respond with a 0, therefore this symbol is chosen for the output to an input of 0 in state BDF. For an input symbol of 1, two of the three states respond with a 0, thus, this being the majority decision, this symbol is chosen for the output to an input of 1 in state BDF. Note that several states of the majority logic machine are isolated from the start state and could never be expressed.

C3.3.5 Crossover: parse trees

Peter J Angeline

Abstract

Described here is the standard crossover operation for parse tree representations most often used in genetic programming. Extensions to this operator for subtrees with multiple return types and genetic programs using automatically defined functions are also described.

From an evolutionary computation view, crossover, in its most basic form, is an operator that exchanges representational material between two parent structures to produce offspring. Occasionally, it is important to introduce additional constraints on the crossover operation to ensure that the created children observe certain necessary constraints of the representation or problem environment.

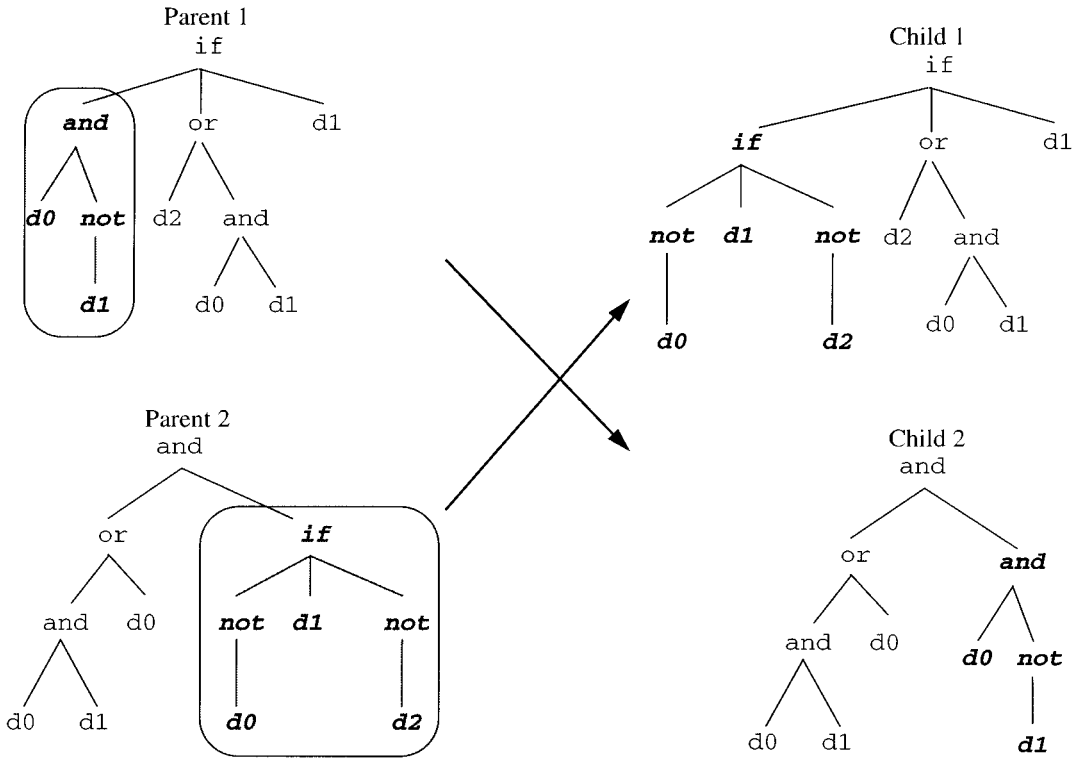


Figure C3.3.10. An illustration of the crossover operator for parse trees. A subtree is selected at random from each parent, extracted, and exchanged to create two offspring trees.

Parse tree representations, as typically used in *genetic programming* (Koza 1992), require that the crossover operation produce offspring that are also valid parse trees. In order to remain a valid parse tree, the structure must have only terminals at the leaf positions of the tree and only function nodes at each of its internal positions. In addition, each function node of the parse tree must have the correct number of subtrees below it, one for each argument that the function requires. C1.6, B1.5.1

Often in genetic programming, a simplification is made so that all functions and terminals in the primitive language return the same data type. This is referred to as the *closure principle* (Koza 1992). The effect is to reduce the number of syntactic constraints on the programs so that the complexity of the crossover operation is minimized.

The recursive structure of parse tree representations makes the definition of crossover for tree representations that adhere to the above caveats surprisingly simple. Cramer (1985) initially defined the now standard subtree crossover for parse trees shown in figure C3.3.10. First, a random subtree is selected and removed from one of the parents. Note that this leaves a hole in the parent such that there exists a function that has a null value for one of its parameters. Next, a random subtree is extracted from the second parent and inserted at the point in the first parent where its subtree was removed. Now the hole in the first parent is again filled. The process is completed by inserting the subtree extracted from the first parent into the position in the second parent where its subtree was removed. As long as only complete subtrees are swapped between parents and the closure principle holds, this simple crossover operation is guaranteed to produce syntactically valid offspring every execution.

Typically, when evolving parse tree representations, a user-defined limit on the maximum size of any tree in the population is provided. Subtree crossover will often increase the size of a given parent such that, over a number of generations, individuals in an unrestricted population may grow to swamp the available computational resources. Given a user-defined restriction on subtree size, expressed as a limit according to either the depth of a tree or the number of nodes it contains, crossover must enforce this limit. When a crossover operation is executed that creates one or more offspring that violate the size limitation, the crossover operation is invalidated and the offspring are restored to their original forms. What happens next is a matter of choice. Some systems will reject both children and revert back to selecting two new

parents. Other systems attempt crossover repeatedly either until both offspring fall within the size limit or until a specified number of attempts is reached. Given the nature of the crossover operation, the likelihood of performing a valid crossover operation in a small number of attempts, say five, is fairly good.

Koza (1992) popularized the use of subtree crossover for manipulating parse tree representations in genetic programming. The subtree swapping crossover of Koza (1992) shares much with the subtree crossover defined by Cramer (1985) with a few minor differences. The foremost difference is a bias introduced by Koza (1992) to limit the probability that a leaf node is selected as the subtree from a parent during crossover. The reasoning for this bias according to Koza (1992) is that, in most trees, the number of leaf nodes will be roughly equivalent to the number of nonleaf nodes. Consequently, the number of subtrees of depth one will be approximately the number of subtrees of depth greater than one. Merely swapping a leaf between parents to produce children half of the time will not tend to greatly advance the evolutionary process, so, during crossover in a genetic program, the probability that a leaf node is selected is controlled by a bias term called the leaf frequency. Typically, the leaf frequency is set at about 10%, meaning that 10% of the time when a subtree is selected a leaf node will be chosen in a parent while the rest of the time only nonleaf nodes will be chosen. Koza (1992) offers no empirical validation of this bias term or its assumed value.

Often it is important to violate the closure principle and allow multiple types in the parse tree representation in order to more effectively solve a given problem. This implies that there are some functions such that they cannot be used as arguments to certain other functions. Crossover in such typed parse trees, as described by Montana (1995), proceeds much as in subtree crossover with one caveat to compensate for the additional constraint of multiple return types. First, a random node is selected in the first parent's parse tree. The return type of the root of the subtree is determined and the selection of crossover points in the second parent is restricted to only those subtrees that have identical return types. This ensures that the syntactic constraints in both parents are upheld.

When evolving genetic programs using *automatically defined functions* (ADFs), Koza (1994) uses a slightly modified version of subtree crossover. When crossing two genetic programs with ADFs, if the crossover position in the first tree is selected to be within a particular subroutine then only crossover points in the corresponding subroutine in the second parent are considered. This is similar to the typed crossover of Montana (1995) except that, rather than restricting the crossover positions in the second parent based on the type of subtree extracted from the first, it restricts the selection using the functional origin of the initially selected subtree.

C3.3.6 Other representations

Peter J Angeline and David B Fogel

Abstract

We briefly consider recombination on mixed-integer representations and data structures incorporating introns.

The use of recombination on the alternative mixed-integer representations, and those using introns, does not generally vary from the standard usage. All of the available options of discrete and intermediate recombination apply to the mixed-integer format offered by Bäck and Schütz (1995). Introns are used with the belief that they will enhance the chances for crossover to recombine building blocks. Moreover, Wu and Lindsay (1995) suggest that the addition of introns can have an equivalent effect of varying crossover probabilities across a chromosome, and state 'the advantages of the noncoding segment method including the fact that the genetic algorithm does not need to be modified to handle variable crossover probabilities and that crossover location calculations are much simpler'.

References

Ackley D H 1987 *A Connectionist Machine for Genetic Hillclimbing* (Boston, MA: Kluwer)

Altenberg L 1995 The schema theorem and Price's theorem *Foundations of Genetic Algorithms 3* ed L Whitley and M Vose (San Mateo, CA: Morgan Kaufmann)

Bäck T, Rudolph G and Schwefel H-P 1993 Evolutionary programming and evolution strategies: similarities and differences *Proc. 2nd Ann. Conf. on Evolutionary Programming (San Diego, CA)* ed D B Fogel and W Atmar (La Jolla, CA: Evolutionary Programming Society) pp 11–22

Bäck T and Schütz M 1995 Evolution strategies for mixed-integer optimization of optical multilayer systems *Proc. 4th Ann. Conf. on Evolutionary Programming (San Diego, CA, March 1995)* ed J R McDonnell, R G Reynolds and D B Fogel (Cambridge, MA: MIT Press) pp 33–51

Birgmeier M 1996 Evolutionary programming for the optimization of trellis-coded modulation schemes *Proc. 5th Ann. Conf. on Evolutionary Programming* ed L J Fogel, P J Angeline and T Bäck (Cambridge, MA: MIT Press) at press

Blanton J and Wainwright R 1993 Multiple vehicle routing with time and capacity constraints using genetic algorithms *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 452–9

Booker L B 1982 *Intelligent Behavior as an Adaptation to the Task Environment* Doctoral Dissertation, Department of Computer and Communication Sciences, University of Michigan

——1987 Improving search in genetic algorithms ed L Davis *Genetic Algorithms and Simulated Annealing* (San Mateo, CA: Morgan Kaufmann)

——1993 Recombination distributions for genetic algorithms ed L Whitley *Foundations of Genetic Algorithms 2* (San Mateo, CA: Morgan Kaufmann)

Bridges C L and Goldberg D E 1987 An analysis of reproduction and crossover in a binary-coded genetic algorithm *Proc. 2nd Int. Conf. on Genetic Algorithms (Cambridge, MA, 1987)* ed J J Grefenstette (Cambridge, MA: Erlbaum) pp 9–13

Christiansen F B 1989 The effect of population subdivision on multiple loci without selection ed M W Feldman *Mathematical Evolutionary Theory* (Princeton, NJ: Princeton University Press)

Cramer N L 1985 A representation for the adaptive generation of simple sequential programs *Proc. 1st Int. Conf. on Genetic Algorithms and Their Applications* ed J J Grefenstette (Hillsdale, NJ: Erlbaum) pp 183–7

Davis L 1985 Applying adaptive algorithms to epistatic domains *Proc. Int. Joint Conf. on Artificial Intelligence*

——1989 Adapting operator probabilities in genetic algorithms *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, June 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 61–9

——(ed) 1991 *Handbook of Genetic Algorithms* (New York: Van Nostrand Reinhold)

De Jong K A 1975 *An Analysis of the Behavior of a Class of Genetic Adaptive Systems* Doctoral Dissertation, Department of Computer and Communication Sciences, University of Michigan

Dzubera J and Whitley D 1994 Advanced correlation analysis of operators for the traveling salesman problem *Parallel Problem solving from Nature—PPSN III (Proc. Int. Conf. on Evolutionary Computation and 3rd Conf. on Parallel Problem Solving from Nature, Jerusalem, October 1994)* (*Lecture Notes in Computer Science 866*) ed Yu Davidor, H-P Schwefel and R Männer (Berlin: Springer) pp 68–77

Eiben A E, Raué P-E and Ruttkay Zs 1994 Genetic algorithms with multi-parent recombination *Parallel Problem solving from Nature—PPSN III (Proc. Int. Conf. on Evolutionary Computation and 3rd Conf. on Parallel Problem Solving from Nature, Jerusalem, October 1994)* (*Lecture Notes in Computer Science 866*) ed Yu Davidor, H-P Schwefel and R Männer (Berlin: Springer) pp 77–87

Eiben A E, van Kemenade C H M and Kok J N 1995 Orgy in the computer: multi-parent reproduction in genetic algorithms *Proc. 3rd. Eur. Conf. on Artificial Life (Lecture Notes in Artificial Intelligence 929)* (Berlin: Springer) pp 934–45

Eshelman L J 1991 The CHC adaptive search algorithm: how to have safe search when engaging in nontraditional genetic recombination ed G Rawlins *Foundations of Genetic Algorithms* (San Mateo, CA: Morgan Kaufmann)

Eshelman L J, Caruana R A and Schaffer J D 1989 Biases in the crossover landscape *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, June 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 10–19

Fogel D B 1995 *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence* (Piscataway, NJ: IEEE)

Fogel L J 1964 *On the Organization of Intellect* Doctoral Dissertation, UCLA

Fogel L J and Fogel D B 1986 *Artificial Intelligence through Evolutionary Programming* Final Report for US Army Research Institute, contract no PO-9-X56-1102C-1

Fogel L J, Owens A J and Walsh M J 1966 *Artificial Intelligence Through Simulated Evolution* (New York: Wiley)

Fox B R and McMahan M B 1991 Genetic operators for sequencing problems *Foundations of Genetic Algorithms* ed G J E Rawlins (San Mateo, CA: Morgan Kaufmann) pp 284–300

Furuya H and Haftka R T 1993 Genetic algorithms for placing actuators on space structures *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 536–42

Geiringer H 1944 On the probability theory of linkage in Mendelian heredity *Ann. Math. Stat.* **15** 25–57

Goldberg D and Lingle R Jr 1985 Alleles, loci, and the traveling salesman problem *Proc. 1st Int. Conf. on Genetic Algorithms (Pittsburgh, PA, July 1995)* ed J J Grefenstette (Hillsdale, NJ: Erlbaum)

Gorges-Schleuter M 1989 ASPARAGOS an asynchronous parallel genetic optimization strategy *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, June 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann)

Grefenstette J 1986 Optimization of control parameters for genetic algorithms *IEEE Trans. Syst. Man Cybern.* **SMC-16** 122–8

Holland J H 1975 *Adaptation in Natural and Artificial Systems* (Ann Arbor, MI: University of Michigan Press)

——1989 Searching nonlinear functions for high values *Appl. Math. Comput.* **32** 255–74

Julstrom B A 1995 What have you done for me lately? Adapting operator probabilities in a steady-state genetic algorithm *Proc. 6th Int. Conf. on Genetic Algorithms (Pittburgh, PA, July 1995)* ed L J Eshelman (San Francisco, CA: Morgan Kaufmann) pp 81–7

Koza J R 1992 *Genetic Programming: on the Programming of Computers by Means of Natural Selection* (Cambridge, MA: MIT Press)

——1994 *Genetic Programming II: Automatic Discovery of Reusable Programs* (Cambridge, MA: MIT Press)

Liepins G and Vose M 1992 Characterizing crossover in genetic algorithms *Ann. Math. Artificial Intell.* **5** 27–34

Michalewicz Z 1996 *Genetic Algorithms + Data Structures = Evolution Programs* 3rd edn (Berlin: Springer)

Michalewicz Z, Nazhiyath G and Michalewicz M 1996 A note on the usefulness of geometrical crossover for numerical optimization problems *Proc. 5th Ann. Conf. on Evolutionary Programming* ed L J Fogel, P J Angeline and T Bäck (Cambridge, MA: MIT Press) at press

Mitchell T M 1980 *The Need for Biases in Learning Generalizations* Technical Report CBM-TR-117, Department of Computer Science, Rutgers University; 1990 Shavlik J and Dietterich T (eds) *Readings in Machine Learning* (San Mateo, CA: Morgan Kaufmann)

Montana D J 1995 Strongly typed genetic programming *Evolutionary Comput.* **3** 199–230

Mühlenbein H 1991 Evolution in time and space—the parallel genetic algorithm *Foundations of Genetic Algorithms* ed G J E Rawlins (San Mateo, CA: Morgan Kaufmann)

Oliver I, Smith D and Holland J 1987 A study of permutation crossover operators on the traveling salesman problem *Proc. 2nd Int. Conf. on Genetic Algorithms (Cambridge, MA, 1987)* ed J J Grefenstette (Hillsdale, NJ: Erlbaum) pp 224–30

Radcliffe N J 1991 Forma analysis and random respectful recombination *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, July 1991)* ed R K Belew and L B Booker (San Mateo, CA: Morgan Kaufmann) pp 222–9

——1994 The algebra of genetic algorithms *Ann. Math. Artificial Intell.* **10** 339–84

Radcliffe N and Surry P D 1995 Fitness variance of formae and performance prediction *Foundations of Genetic Algorithms 3* ed D Whitley and M Vose (San Mateo, CA: Morgan Kaufmann) pp 51–72

Reed J, Toombs R and Barricelli N A 1967 Simulation of biological evolution and machine learning *J. Theor. Biol.* **17** 319–42

Renders J-M and Bersini H 1994 Hybridizing genetic algorithms with hill-climbing methods for global optimization: two possible ways *Proc. 1st IEEE Conf. on Evolutionary Computation (Orlando, FL, June 1994)* (Piscataway, NJ: IEEE) pp 312–7

Robbins R B 1918 Some applications of mathematics to breeding problems, III *Genetics* **3** 375–89

Schaffer J D, Caruana R A, Eshelman L J and Das R 1989 A study of control parameters affecting online performance of genetic algorithms for function optimization *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, June 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 51–60

Schaffer J D and Morishima A 1987 An adaptive crossover distribution mechanism for genetic algorithms *Proc. 2nd Int. Conf. on Genetic Algorithms (Cambridge, MA, 1987)* ed J J Grefenstette (Hillsdale, NJ: Erlbaum) pp 36–40

Schnell F W 1961 Some general formulations of linkage effects in inbreeding *Genetics* **46** 947–57

Spears W M and De Jong K A 1991a An analysis of multi-point crossover ed G Rawlins *Foundations of Genetic Algorithms* (San Mateo, CA: Morgan Kaufmann)

——1991b On the virtues of parameterized uniform crossover *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, July 1991)* ed R K Belew and L B Booker (San Mateo, CA: Morgan Kaufmann) pp 230–6

Srinivas M and Patnaik L M 1994 Adaptive probabilities of crossover and mutation in genetic algorithms *IEEE Trans. Syst. Man Cybernet.* **SMC-24** 656–67

Syswerda G 1989 Uniform crossover in genetic algorithms *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, June 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 2–9

——1991 Schedule optimization using genetic algorithms *Handbook of Genetic Algorithms* ed L Davis (New York: Van Nostrand Reinhold) pp 332–49

Thierens D and Goldberg D E 1993 Mixing in genetic algorithms *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 38–45

Whitley D and Yoo N-W 1995 Modeling simple genetic algorithms for permutation problems *Foundations of Genetic Algorithms 3* ed D Whitley and M Vose (San Mateo, CA: Morgan Kaufmann) pp 163–84

Whitley D, Starkweather T and Shaner D 1991 Traveling salesman and sequence scheduling: quality solutions using genetic edge recombination *Handbook of Genetic Algorithms* ed L Davis (New York: Van Nostrand Reinhold) pp 350–72

Wright A H 1994 Genetic algorithms for real parameter optimization *Foundations of Genetic Algorithms* ed G Rawlins (San Mateo, CA: Morgan Kaufmann) pp 205–18

Wu A S and Lindsay R K 1995 Empirical studies of the genetic algorithm with noncoding segments *Evolutionary Comput.* **3** 121–48

Zhou H and Grefenstette J J 1986 Induction of finite automata by genetic algorithms *Proc. 1986 IEEE Int. Conf. on Systems, Man, and Cybernetics (Atlanta, GA)* pp 170–4

C3.4 Other operators

Russell W Anderson (C3.4.1), *David B Fogel* (C3.4.2) *and Martin Schütz* (C3.4.3)

Abstract

See the individual abstracts for sections C3.4.1–C3.4.3.

C3.4.1 The Baldwin effect

Russell W Anderson†

Abstract

The Baldwin effect is a passive evolutionary process, whereby individual learning facilitates genetic evolution. Baldwinian evolution is distinguished from the more active (and nonbiological) Lamarckian inheritance of acquired characters. The principles underlying the Baldwin effect are explained and its manifestations in evolutionary algorithms are discussed. A first-order analysis using quantitative genetics is used to illustrate some common misconceptions. When appropriately implemented, hybrid algorithms can efficiently exploit the Baldwin effect in evolutionary optimization.

C3.4.1.1 Interactions between learning and evolution

In the course of an evolutionary optimization, solutions are often generated with low phenotypic fitness even though the corresponding genotype may be close to an optimum. Without additional information about the local *fitness landscape*, such genetic *near misses* would be overlooked under strong selection. Presumably, one could rank near misses by performing a local search and scoring them according to distance from the nearest optimum. Such evaluations are essentially the goal of hybrid algorithms (Chapter B1.5, Balakrishnan and Honavar 1995), which combine global search using evolutionary algorithms and local search using individual learning algorithms. Hybrid algorithms can exploit learning either actively (via Lamarckian inheritance) or passively (via the Baldwin effect). B2.7

Under Lamarckian algorithms, performance gains from individual learning are mapped back into the genotype used for the production of the next generation. This is analogous to Lamarckian inheritance in evolutionary theory—whereby characters acquired during a parent's lifetime are passed on to their offspring. Lamarckian inheritance is rejected as a biological mechanism under the modern synthesis, since it is difficult to envision a process by which acquired information can be transferred into the gametes. Nevertheless, the practical utility of Lamarckian algorithms has been demonstrated in some evolutionary optimization applications (Ackley and Littman 1994, Paechter *et al* 1995). Of course, these algorithms are limited to problems where a reverse mapping from the learned phenotype to genotype is possible.

However, even under purely Darwinian selection, individual learning influences evolutionary processes, but the underlying mechanisms are subtle. The 'Baldwin effect' is one such mechanism, whereby learning facilitates the assimilation of new genetic innovations (Baldwin 1896, Morgan 1896, Osborn 1896,

† This work was supported by the Public Health Foundation and the Kett Foundation. The author wishes to thank David Fogel and Peter Turney for encouragement and comments.

Waddington 1942, Hinton and Nowlan 1987, Maynard Smith 1987, Anderson 1995a, Turney *et al* 1996). Learning allows an individual to complete and exploit partial genetic programs and thereby survive. In other words, learning guides evolution by assigning 'partial credit' for genetic near misses. Individuals with useful genetic variations are thus maintained by learning, and the corresponding genes increase in frequency in the subsequent generation. As genetic components necessary for a complex structure accumulate in the gene pool, functions that previously required supplemental learning are replaced by genetically determined systems.

Empirical studies can quantify the benefits of incorporating individual learning into evolutionary algorithms (Belew 1989, French and Messinger 1994, Nolfi *et al* 1994, Whitley *et al* 1994, Cecconi *et al* 1995). However, a theoretical treatment of the effects of learning on evolution can strengthen our intuition for *when* and *how* to implement such approaches. This section presents an overview of the principles underlying the Baldwin effect, beginning with a brief history of the elucidation and development in evolutionary biology. Computational models of the Baldwin effect are reviewed and critiqued. The Baldwin effect is then analyzed using standard quantitative genetics. Given reasonable assumptions of the effects of learning on fitness and its associated costs, this theoretical approach builds and strengthens conventional intuition about the effects of individual learning on evolution. Finally, issues concerning problem formulation, learning algorithms, and algorithmic design are discussed.

C3.4.1.2 The Baldwin effect in evolutionary biology

Complex biological structures require the coordinated expression of several genes in order to function properly. Determining how such structures arise through evolution is problematic because it is often difficult to envision the evolutionary advantage offered by intermediate forms. Without additional developmental mechanisms, individuals with incomplete genetic programs would gain no evolutionary advantage over those devoid of any genetic components.

Baldwin (1896), Osborn (1896) and Morgan (1896) proposed how individual learning can facilitate the evolution of complex genetic structures by protecting partial genetic innovations, or 'ontogenetic variations': '[learning] supplements such partial co-ordinations, makes them functional, and so keeps the creature alive' (Baldwin 1896). Baldwin further proposed how this individual advantage of learning guides the process of evolution: 'the variations which were utilized for ontogenetic adaptation in the earlier generation, being thus kept in existence, are utilized more widely in the subsequent generation' (Baldwin 1896). Over evolutionary time, abilities that were previously maintained by adaptive systems can be replaced by genetically determined systems (i.e. instincts). Waddington proposed an analogous interaction between developmental processes and evolution, whereby developmental adaptations 'guide' or 'canalize' evolutionary change (Waddington 1942, Hinton and Nowlan 1987). Formal mathematical or analytical models quantifying the Baldwin effect did not appear in the literature until fairly recently.

Hinton and Nowlan's model. The first quantitative model demonstrating the Baldwin effect was constructed by Hinton and Nowlan (1987). They used a computer simulation to study the effects of individual learning on the evolution of a population of *neural networks*. They considered an extremely D1
difficult problem, where a network conferred a fitness advantage only if it was fully functioning (all connections wired correctly). Each network was given 20 possible connections, specified by 20 genes.

Briefly consider the difficulty of finding this solution using a pure genetic algorithm. Under a binary genetic coding scheme (allelic values of either 'correct' or 'incorrect'), the probability of randomly generating a functional net is 2^{20}. Note that a net with 19 out of 20 correct connections is no better off than one with no correct connections. The corresponding fitness landscape has a singularity at the correct solution with no useful gradient information, analogous to a putting green (figure C3.4.1). Finding this solution by a pure genetic algorithm, then, is the evolutionary equivalent of a 'hole in one'. Of course, given a large enough random population, an evolutionary algorithm could theoretically find this solution in one generation.

Hinton and Nowlan modeled a modified version of this problem, where genes were allowed *three* alternative forms (alleles): present (1), absent (0), or 'plastic' (?). Connections specified by plastic alleles could be varied by random trials during the individual's life span. This allowed an individual to complete and exploit a partially hard-wired network. Hence, genetic near misses (e.g. 19 out of 20 correct genes) could quickly learn the remaining connection(s) and differentially survive. The presence of plastic alleles, therefore, softened the fitness landscape (figure C3.4.1). Hinton and Nowlan described the effect of learning

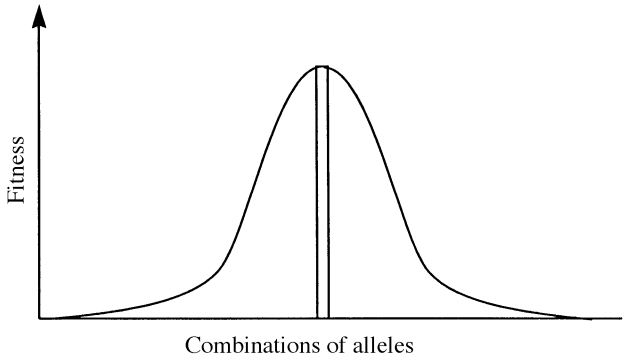


Figure C3.4.1. Schematic representation of the fitness landscape in the model of Hinton and Nowlan. A two-dimensional representation of genome space in the problem considered by Hinton and Nowlan (1987). The horizontal axis represents all possible gene combinations, and the vertical axis represents relative fitness. Without learning, only one combination of alleles correctly completes the network; hence only one genotype has higher fitness, and no gradient exists. The presence of plastic alleles radically alters this fitness landscape. Assume a correct mutation occurs in one of the 20 genes. The advent of a new correct gene only partially solves the problem. Learning allows individuals close (in Hamming space) to complete the solution. Thus, these individuals will be slightly more fit than individuals with no correct genes. Useful genes will thereby be increased in subsequent generations. Over time, a large number of correct genes will accumulate in the gene pool, leading to a completely genetically determined structure.

ability in their simulation as follows: '[learning] alters the shape of the search space in which evolution operates and thereby provides good evolutionary paths towards sets of co-adapted alleles'. The second aspect of the Baldwin effect (genetic assimilation) was manifested in the mutation of plastic alleles into genetically fixed alleles.

Issues raised with computational models. Hinton and Nowlan's paper is regarded as a landmark contribution to understanding the interactions between learning and evolution (Mitchell and Belew 1995) and has inspired a proliferation of modeling studies (Fontanari and Meir 1990, Ackley and Littman 1991, 1994, Whitley and Gruau 1993, Whitley *et al* 1994, Balakrishnan and Honavar 1995, Turney 1995, 1996, Turney *et al* 1996). Considering the rather specific assumptions of their model, it is useful to contemplate which aspects of their results are general properties. Among the issues raised by this and subsequent studies are the degree of biological realism, the nature of the fitness landscape, the computational cost of learning, and the role of learning in static fitness landscapes.

First, the model's assumption of plastic alleles that can mutate into permanent alleles seems biologically spurious. However, the Baldwin effect can be manifested in the evolution of a biological structure regardless of the genetic basis of that structure or the mechanisms underlying the learning process (Anderson 1995a). The Baldwin effect is simply a consequence of individual learning on genetic evolution. Subsequent studies have demonstrated the Baldwin effect using a variety of learning algorithms. Turney (1995, 1996) has observed a Baldwin effect in a class of hybrid algorithms, combining a genetic algorithm (GENESIS) and an inductive learning algorithm, where the Baldwin effect was manifested in shifting biases in the inductive learner. French and Messinger (1994) investigated the Baldwin effect under various forms of phenotypic plasticity. Cecconi *et al* (1995) observed the Baldwin effect in a *GA+NN hybrid* (a hybrid D1
of a genetic algorithm and a neural network), as did Nolfi *et al* (1994) and Whitley and Gruau (1993). Unemi *et al* (1994) demonstrated the Baldwin effect in a GA+RL hybrid (GA and reinforcement learning; in particular, they studied Q-learning). Whitley *et al* (1994) studied the Baldwin effect with a hybrid of a GA and a simple hill climbing algorithm. Finally, it is interesting to note that genetic mechanisms closely analogous to the plastic alleles of Hinton and Nowlan may be in effect in evolutionary interactions between natural and adaptive antibodies (Anderson 1995b, 1996a). Nevertheless, it is difficult to see how this particular model could be generalized to learning in neural systems.

Second, the model of Hinton and Nowlan assumed an extremely rugged fitness landscape. The assumption of an 'all-or-nothing' fitness landscape has apparently led some to assert that a nonlinear selection function is *necessary* for a Baldwin effect to occur (Hightower *et al* 1996). This claim is not supported by rigorous analysis. Learning can alter the shape of *any* fitness landscape and therefore can

affect evolutionary trajectories. For example, consider linear directional selection. If learning only serves to change the *slope* of the selection function, it will by definition affect its severity.

Third, the observation that 'learning *facilitates* evolution,' has often been interpreted as 'learning *accelerates* evolution'. Although several empirical studies have demonstrated increased convergence rates for hybrid algorithms (Parisi *et al* 1991, Turney 1995, Ackley and Littman 1991, 1994, Balakrishnan and Honavar 1995), this more general claim is untenable under many conditions. Intuitively, learning can *slow* genetic change by protecting otherwise less optimal genotypes from selection. Furthermore, individual adaptive abilities can represent an enormous investment of resources (consider the cerebral cortex in man!). Since individual learning accrues a computational or biological cost, the *costs and benefits* of learning must be weighed before drawing such conclusions.

Fourth, most current hybrid algorithm applications operate on a fixed problem, or static fitness landscape. An exception is a study by Unemi *et al* (1994), which involves a simulated robot in a maze. They show that the ability to learn is initially beneficial, but it will eventually be selected out of the gene pool, unless the maze changes dynamically with each new individual trial. Ultimately, learning has no selective advantage in fixed environments, since, presumably, once the optimal genotype is found, exploration away from this optimum only reduces fitness (Stephens 1993, Via 1993, Anderson 1995a). The studies by Hinton and Nowlan (1987) and Fontanari and Meir (1990) corroborate this thesis: their simulations showed that as individuals arose with allelic combinations close to the optimum, the plastic alleles (representing the ability to learn) were selected out of the gene pool. In other words, the computational advantage of individual learning decreases over the course of an evolutionary optimization. Under these conditions, individual learning can only be maintained in a population subject to changing environmental conditions. A similar case has been made for phenotypic plasticity in general (West-Eberhard 1989, Stearns 1989, Scheiner 1993, Via 1993) as well as for sexual versus asexual reproduction (Maynard-Smith 1978).

C3.4.1.3 Quantitative genetics models

In order to make some of these issues more explicit, it is useful to study the Baldwin effect under the general assumptions of quantitative genetics. A quantitative genetics methodology for modeling the effects of learning on evolution was developed by Anderson (1995a), and the primary results of this analysis are reviewed in this section. The limitations of this theoretical approach are well known. For example, quantitative genetics assumes infinite population sizes. Also, complete analysis is often limited to a single quantitative character. Nevertheless, such analyses can provide a baseline intuition regarding the effects of learning and evolution.

All essential elements of an evolutionary process subject to the Baldwin effect are readily incorporated into a quantitative genetics model. These elements include (i) a function for the generation of new genotypes through mutation and/or recombination, (ii) a mapping from genotype to phenotype, (iii) a model of the effects of learning on phenotype, and (iv) a selection function. In this section, this methodology is demonstrated for a simple, first-order model, where only the phenomenological *effects* of learning on selection are considered. More advanced models are discussed, which incorporate a model of the learning process, along with its associated costs and benefits. These analyses illustrate several underappreciated points: (i) learning generally slows genetic change, (ii) learning offers no long-term selective advantage in fixed environments, and (iii) the effects of learning are somewhat independent of the mechanisms underlying the learning process.

Learning as a phenotypic variance. For a first-order model, consider an individual whose genotype is a real-valued quantitative character subject to normal (Gaussian) selection:

$$w_s(g) \sim N(g_e, V_s) \tag{C3.4.1}$$

where $w_s(g)$ represents selection as a function of genotype, g_e represents the optimal genotype, and $V_s(t)$ is variance of selection as a function of time. A direct mapping from genotype to phenotype is implicitly assumed.

What effect does learning have on this selection function? Learning allows an individual to modify its phenotype in response to its environment. Consider an individual whose genotype (g_i) is a given distance ($|g_i - g_e|$) from the environmental optimum (g_e). Regardless of the mechanisms underlying the learning

process, the net effect of learning is to reduce the fitness penalty associated with this genetic distance. Because of its ability to learn, an individual with genotype g_i has a probability of modifying its phenotype to the environmental value g_e which is a function of the distance between these two values. A simple way to model this effect is to specify a phenotypic variance due to learning (V_l). This is equivalent to increasing the variance of selection. Thus, learning increases the width of the selection function such that V_s is replaced by $V_s^* = V_s + V_l$.

Fixed selection, constant learning. Consider a population subject to selection with a fixed environmental optimum. For simplicity, let $g_e = 0$. Assume an initial Gaussian distribution of genotypes, $f_p(g) = N(m(t), V_p(t))$, where $m(t)$ and $V_p(t)$ are the population mean and variance at time t. Each round of selection changes the distribution of genotypes according to

$$f_p^*(g) \sim f_p(g) w_s(g) \tag{C3.4.2}$$

$$\sim \exp\left\{-\tfrac{1}{2}[(g - m(t))^2/V_p(t) + g^2/V_s^*]\right\} \Rightarrow N(m, V_p) * N(0, V_s) \tag{C3.4.3}$$

$$\sim \exp\left\{-\tfrac{1}{2}[(g - m^*(t))^2/V_p^*(t)]\right\} \Rightarrow N(m^*(t), V_p^*(t)). \tag{C3.4.4}$$

The population mean and variance after selection (m^*,V_p^*) can now be expressed in the form of dynamic equations:

$$m^*(t) = \frac{m(t)V_s^*}{V_p(t) + V_s^*} = m(t) - \frac{m(t)V_p(t)}{V_p(t) + V_s^*} \tag{C3.4.5}$$

$$V_p^*(t) = \frac{V_p(t)V_s^*}{V_p(t) + V_s^*} = V_p(t) - \frac{V_p^2(t)}{V_p(t) + V_s^*}. \tag{C3.4.6}$$

Lastly, mutations are introduced in the production of the next generation of trials. To model this process, assume a Gaussian mutation function with mean zero and variance V_μ. A convolution of the population distribution with the mutation distribution has the effect of increasing the population variance:

$$f_p^{**}(g) = \int_{-\infty}^{+\infty} f_p^*(s) f_m(g - s)\,\mathrm{d}s = N(m^*(t), V_p^{**}(t)) \tag{C3.4.7}$$

where

$$V_p^{**}(t) = V_p(t) - \frac{V_p^2(t)}{V_p(t) + V_s^*} + V_\mu. \tag{C3.4.8}$$

Hence, in a fixed environment the population mean $m(t)$ will converge on the optimal genotype (Bulmer 1985), while a mutation–selection equilibrium variance occurs at

$$V_{peq}^{**} = \frac{V_\mu + (V_\mu^2 + 4V_\mu V_s^*)^{1/2}}{2}. \tag{C3.4.9}$$

Inspection of equations (C3.4.5), (C3.4.6), and (C3.4.8) illustrates two important points. First, learning slows the convergence of both $m^*(t)$ and $V_p^*(t)$. Second, once convergence in the mean is complete, the utility of learning is lost, and learning only reduces fitness.

In a more elaborate version of this model, called the *critical learning period model* (Anderson 1995a), a second gene is introduced to regulate the fraction of an individual's life span devoted to learning (duration of the learning period). Specification of a critical learning period implicitly assigns cost associated with learning (the percent of life span not devoted to reproduction). Individuals are then selected for the optimal combination of genotype and learning investment. It is easily demonstrated that under these assumptions, learning ability is selected out of a population subject to fixed selection.

Constant-velocity environments. Next, consider a simple case of changing selection—a constantly moving optimum, $g_e(t) = \delta t$, where δ is defined as the environmental velocity. Let the difference between the population mean and the environmental optimum be defined as $\phi = m(t) - g_e(t)$. The dynamic equation for ϕ is

$$\phi^*(t) = \phi(t) - \frac{\phi(t)V_p(t)}{V_p(t) + V_s^*} + \delta. \tag{C3.4.10}$$

At equilibrium, $\phi^*(t) = \phi(t)$, hence

$$\phi_{\text{eq}} = \frac{V_{\text{p}} + V_{\text{s}}^*}{V_{\text{p}}}\delta \tag{C3.4.11}$$

where the equilibrium is expressed as a distance from the optimum. A similar result can be found in the article by Charlesworth (1993), in his analysis of the evolution of sex in a variable environment. The equilibrium population variance remains the same as in the case of a fixed environment. Substituting (C3.4.9) yields

$$\phi_{\text{eq}} = \frac{\delta}{2}\left[(1 + (1 + 4V_{\text{s}}^*/V_{\mu})^{1/2}\right]. \tag{C3.4.12}$$

Thus in an environment where the optimal phenotype is changing at constant rate, the population mean genotype converges on a constant 'phase lag' (ϕ_{eq}).

Learning actually increases the phase lag by protecting suboptimal genotypes from selection. But this model assumes $\bar{w}_{\text{s}} = 1$, so that only the relative magnitude of selection is accounted for. Strong selection without learning might actually lead to *extinction* in rapidly changing selection. Phenotypic variability (due to learning) has the effect of 'shielding' these marginal genotypes from selection (Wright 1931).

As environmental conditions change, so will the selective advantage of learning. The relations derived in this analysis show which equilibria will be reached for an assumed phenotypic variability, but the model does not yield information on what would represent the optimal investment in learning. Hence, a complete model of the benefits of the Baldwin effect must incorporate the costs associated with learning. The best way to estimate these costs is to develop a model of the underlying learning process.

Models of learning. A reasonable question to ask is how sensitive are the effects of learning to the mechanisms underlying the learning process. The most direct (and exhaustive) method for investigating this question would be to construct a computer simulation to compare the effects of two learning processes in an evolutionary program. However, estimates of comparative performance can also be obtained using quantitative genetics models according to the following methodology. First, one must develop a model of the learning process. Next the effects of the learning process must be mapped onto the selection function. A simple approximation is to construct a probabilistic or phenomenological model of the effect of learning on phenotype.

Under the critical learning period model (Anderson 1995a), learning consists of a series of independent trials conducted over a fraction of the individual's life span, or learning period. This simple model incorporates two important considerations: the sequential nature of learning and a model of the cost associated with learning. Despite the more complicated assumptions, the dynamical response of this model to various forms of selection (fixed, random variation, and constant velocity) were qualitatively comparable to those derived for the simple additive variance model analyzed here. Longer learning periods increase the investment in (and cost of) learning: consequently, the amount of learning investment generally only increased with increased environmental variability.

Other models of the learning process can be incorporated using the methodology outlined above. For example, under the critical learning period model, individuals were not allowed to benefit from successive trials within the learning period, nor were they allowed to begin exploitation of successful trials until after the learning period. Removing these two restrictions yields a *sequential* trial-and-error learning rule. Such a learning rule is a more appropriate model of the learning process in some systems, such as affinity maturation in the antibody immune system (Milstein 1990) or skill acquisition in neural systems (Bremermann and Anderson 1991, Anderson 1996b). For these initial models, including such details of individual learning was unwarranted, but any model of learning can be mapped onto a fitness function, although mapping a sequential trial-and-error learning rule onto a survival probability may be analytically more difficult. Often it turns out that this mapping masks the details of the underlying process (Anderson 1995a). This suggests that the *effects* of individual learning on evolution will be qualitatively the same.

C3.4.1.4 Conclusions

Baldwin's essential insight was that if an organism has the ability to learn, it can exploit genes that only partially determine a structure—increasing the frequencies of useful genes in subsequent generations. The Baldwin effect has also been demonstrated to be operative in hybrid evolutionary algorithms. These empirical investigations can be used to quantify the benefits of incorporating individual learning into

an evolutionary algorithm. Computation time is the obvious performance criterion; however, such comparisons are often limited to the particular application. Alternatively, phenomenological models can be used to generate reasonable estimates of performance expectations, deferring the arduous task of creating detailed computer simulations.

The introduction of individual learning can radically alter fitness landscapes. This is especially true if the learning algorithm operates on phenotypes according to a fundamentally different process. Clearly, if the learning algorithm is identical to the genetic algorithm, no computational savings are likely to be manifest.

Under certain conditions, learning slows genetic change by protecting suboptimal genotypes from selection. Thus, the benefits of individual learning will probably be accrued early in optimization, when the population is far from equilibrium, and learning will eventually impede algorithmic convergence. Accordingly, for optimizations on fixed fitness landscapes, a 'variable-learning-investment' strategy—where the computational resources applied toward learning are subject to change—should be considered (Saravanan *et al* 1995, Anderson 1995a).

C3.4.2 Knowledge-augmented operators

David B Fogel

Abstract

Incorporating domain-specific knowledge into an evolutionary algorithm can improve the effectiveness and efficiency of the search. Some examples of knowledge-augmented operators are provided.

Evolutionary computation methods are broadly useful because they are general search procedures. The canonical forms of the evolutionary algorithms do not take advantage of knowledge concerning the problem at hand. For example, in the canonical genetic algorithm (Holland 1975), a one-point crossover operator is suggested with a crossover point to be chosen randomly across the parents' chromosomes. However it is generally accepted that the effectiveness of a particular search operator depends on at least three interrelated factors: (i) the chosen representation, (ii) the selection criterion, and (iii) the objective function to be minimized or maximized, subject to the given constraints if applicable. There is no single best search operator for all problems.

Rather than rely on simple operators that may generate unacceptably inefficient performance on a particular problem at hand, the search operators can be tailored for individual applications. For example, in evolution strategies and evolutionary programming, when searching for the minimum of a quadratic surface, Rechenberg (1973) showed that the best choice for the standard deviation σ when using a zero mean Gaussian mutation operator was

$$\sigma = 1.224 f(\boldsymbol{x})^{1/2}/n$$

where $f(\boldsymbol{x})$ is the quadratic function evaluated at the parent vector $\boldsymbol{x}$, and n is the dimensionality of the function. This choice of σ incorporates knowledge about the function being searched in order to provide the greatest expected rate of convergence. In this particular case, however, knowledge that the function is a quadratic surface indicates the use of search algorithms that can take greater advantage of the available gradient information (e.g. Newton–Gauss).

There are other instances where incorporating domain-specific knowledge into a search operator can improve the performance of an evolutionary algorithm. In the *traveling salesman problem*, under the objective function of minimizing the Euclidean distance of the circuit of cities, and a representation of simply an ordered listing of cities to be visited, Fogel (1988) offered a mutation operator which selected a city at random and placed it in the list at another randomly chosen position. This operator was not based on any knowledge about the nature of the problem. In contrast, Fogel (1993) offered an operator that instead inverted a segment of the listing (i.e. like a 2-opt of Lin and Kernighan (1976)). The inversion operator in the traveling salesman problem is a knowledge-augmented operator because it was devised to take advantage of the Euclidean geometry present in the problem. In the case of a traveling salesman's tour, if the tour crosses over itself it is always possible to improve the tour by undoing the crossover (i.e. the diagonals of a quadrangle are always longer in sum than any two opposite sides). When the two

G9.5

cities just before and after the crossing point are selected and the listing of cities in between reversed, the crossing is removed and the tour is improved. Note that this use of inversion is appropriate in light of the traveling salesman problem, and no broader generality of its effectiveness as an operator is suggested, or can be defended.

Domain knowledge can also be applied in the use of recombination. For example, again when considering the traveling salesman problem, Grefenstette *et al* (1985) suggested a heuristic crossover operator that could perform a degree of local search. The operator constructed an offspring from two parents by (i) picking a random city as the starting point, (ii) comparing the two edges leaving the starting cities in the parents and choosing the shorter edge, then (iii) continuing to extend the partial tour by choosing the shorter of the two edges in the parents which extend the tour. If a cycle were introduced, a random edge would be selected. Grefenstette *et al* (1985) noted that offspring were on average about 10% better than the better parent when implementing this operator.

In many real-world applications, the physics governing the problem suggests settings for search parameters. For example, in the problem of docking small molecules into protein binding sites, the intermolecular potential can be precalculated on a grid. Gehlhaar *et al* (1995) used a grid of 0.2 Å, with each grid point containing the summed interaction energy between an atom at that point and all protein atoms within 6 Å. This suggests that under Gaussian perturbations following an evolutionary programming or evolution strategy approach, a standard deviation of several ångströms would be inappropriate (i.e. too large).

Whenever evolutionary algorithms are applied to specific problems with the intention of generating the best available optimization performance, knowledge about the domain of application should be considered in the design of the search operators (and the representation, selection procedures, and indeed the objective function itself).

C3.4.3 Gene duplication and deletion

Martin Schütz

Abstract

A short historical overview referring to gene duplication and deletion is given and four basic motivations for using these concepts in the context of EAs are presented. After a formal description of the duplication and deletion operator, some problems concerning their use are listed. Finally, some ways of solving these problems are explained.

C3.4.3.1 Historical review

The idea of using operators such as gene duplication and deletion in the context of evolutionary algorithms (EAs) is as old as the algorithms themselves.

Fogel *et al* (1966) seemed to be one of the first experimenting with variable-length genotypes. In their work they evolved finite-state machines of a varying number of states, therefore making use of operators such as addition and deletion. Typically, the 'add a state' operator was performed randomly, rather than a strict duplication. They also suggested a 'majority logic' operator that essentially created a machine in which each state was the composite of a state from each of the original machines; that is, this operator duplicated the majority logic vote at each state of multiple finite-state machines. Concerning engineering problems Schwefel (1968) was one of the first using gene duplication and deletion for solving the task of determining the internal shape of a two-phase jet nozzle with maximum thrust under constant starting conditions. Holland (1975, p 111) proposed the concepts of gene duplication and gene deletion in order to raise the computational power of EAs.

C3.4.3.2 Basic motivations for the use of gene duplication and deletion

From these first attempts concerning variable-length genotypes until now many researchers have made use of gene duplication and deletion. Four different motivations may be classified.

(i) *Engineering applications.* Many difficult optimization tasks arise from engineering applications in which variable-dimensional mixed-integer problems have to be solved. Often these problems are of dynamic nature: the optimum is time dependent. Additionally, in order to obtain a reasonable model of the system under consideration, a large number of constraints has to be respected during the optimization. Solving the given task frequently assumes the integration of expert (engineer) knowledge into the problem solving strategy: into particular genetic operators in the case of EAs. Many such constrained, variable-dimensional, mixed-integer, time-varying engineering problems and their solutions can be found in the handbook by Davis (1991) and in the proceedings of several conferences, such as the International Conference on Evolutionary Computation (ICEC), Conference on Genetic Algorithms and Their Applications (ICGA), Conference on Evolutionary Programming (EP) and Parallel Problem Solving from Nature (PPSN). G3

(ii) *Raising the computational power of EAs.* As Goldberg *et al* (1989, p 493; see also Goldberg *et al* 1990) state, 'nature has formed its genotypes by progressing from simple to more complex life forms', thereby using variable-length genotypes. He states that genetic algorithms (GAs) using variable-length genotypes, thus being able to use duplication and deletion operators,

> solve problems by combining relatively short, well-tested building blocks to form longer, more complex strings that increasingly cover all features of a problem. ... Specifically, and more positively, we assert that allowing more messy strings and operators permits genetic algorithms to form and exploit tighter, higher performance building blocks than is possible with random, fixed codings and relatively slow reordering operators such as inversion.

Transferring this idea to EAs in general hopefully leads to more efficient EAs.

(iii) *Extradimensional bypass.* One additional motivation underpinning the usefulness of variable-dimensional genotype lengths is given by the *extradimensional bypass thesis* of Conrad (1993) (given more formally by Conrad and Ebeling (1992)), which states (maximization):

> As the number of dimensions increases the chance of our sitting on top of an isolated peak decreases, assuming that the space has random topographic features. The peaks will be transformed to saddlepoints. The rate of evolution will then depend on how long it takes to discover an uphill running pathway that requires a series of short steps and not on how long it takes to make a long jump from one peak to another.

For example, imagine an alpinist walking in a two-dimensional environment standing in front of a crater whose top he would like to reach. Even if he cannot see the highest peak, climbing the crater and walking on the border of the crater (extradimensional bypass) will lead him to the top. Walking in a one-dimensional space would complicate the task. This time the surface consists of two separated peaks (cut through the crater). If the alpinist climbs the first peak (eventually the higher one) he sees the highest peak but since one dimension is lost the desired path along the borderline of the crater does not exist. This time the alpinist has to descend into the valley in order to solve his task. As one can recognize, introducing extra dimensions during the course of evolution may overcome the problem of becoming stuck in a local optimum or, to put it in other words, decrease the necessity of jumping from one basin of attraction to another.

(iv) *Artificial intelligence.* Another important field in which variable-dimensional techniques have also been used is the domain of artificial intelligence (AI), especially *machine learning* (ML) and *artificial life* (AL). Whereas in the field of ML (subordinated fields are, for example, *genetic programming*, *classifier systems*, and *artificial neural networks*) solving a possibly variable-dimensional optimization problem (depending on the actual subordinated field in mind) is one main objective, this aim plays a minor role in the AL field. AL research concentrates on computer simulations of simple hypothetical life forms and selforganizing properties emerging from local interactions within a large number of basic agents (life forms). A second objective of AL is the question of how to make the agents' behavior adaptive, thus often leading to agents equipped with internal rules or strategies determining their behavior. In order to learn/evolve such rule sets, learning strategies such as EAs are used. Since the number of necessary rules B1.5.1, B1.5.2 D1

is not known *a priori*, a variable-dimensional problem instance arises. Despite the rule learning task, the modeling of the simple life forms itself makes use of variable-dimensional genotypes.

C3.4.3.3 Formal description of gene duplication and deletion

From the preceding motivations one can see that solving variable-dimensional optimization problems with constraints forms one main task forcing the use of gene duplication and deletion. This sort of optimization (minimization) problem may be formalized as follows.

Definition C3.4.1 (variable-dimensional minimization problem with constraints).

Given $f : D \subseteq X = \cup_{i=1}^{\infty} G^i \rightarrow \mathbb{R}$, minimize $f(\boldsymbol{x})$ subject to

$$\begin{aligned} & g_i(\boldsymbol{x}) \geq 0 \qquad \forall i \in \{1, \ldots, m\} \\ & h_j(\boldsymbol{x}) = 0 \qquad \forall j \in \{1, \ldots, l\} \\ & \boldsymbol{x} = (x_1, \ldots, x_{n_x}) \in D \subseteq X \qquad n \in \mathbb{N} \text{ arbitrary but not fixed,} \\ & f, g_i, h_j : X \rightarrow \mathbb{R}. \end{aligned}$$

□

g_i are called *inequality constraints* and h_j *equality constraints*. The main difference concerning a non-variable-dimensional optimization (minimization) problem is the fact that the dimension of the objective vector $\boldsymbol{x}$ may vary; that is, it is not fixed. As a consequence the parameter space X has to be formulated as the union of all parameter spaces G^i of fixed sizes i. In the context of EAs the gene space G ought not to be a vector space as usual in classical optimization (most often a Banach space, e.g. $\mathbb{R}^n$), thereby omitting all the comfortable properties Banach spaces have with respect to analysis. Instead, G might be $\mathbb{B}, \mathbb{N}, \mathbb{Z}, \mathbb{Q}, \mathbb{C}, \mathbb{R}$ or any other complex space (not in the strict mathematical sense) representable by a complex data structure. The use of G is necessary because most duplication and deletion operators directly work on semantical entities represented by G. Davidor (1991a), for example, uses binary encoded vectors of triples (three angles) for representing a robot trajectory, thus G takes the form $G = \mathbb{B}^l \times \mathbb{B}^l \times \mathbb{B}^l$.

Until now we have presented motivations for the use of variable-length genotypes in the field of EAs. Unfortunately, nature gives no real hint at why using a variable-length genotype should be advantageous. A high degree of genepool diversity and a high flexibility to a changing environment may be one main benefit of non-fixed gene lengths, thus raising the evolutional power/adaptability of a population. One interesting fact nature offers is that gene duplication most often leads to viable individuals, whereas gene deletion does not (Futuyma 1986, p 78). (A brief and sufficient introduction into the concepts of *neo-Darwinism*, i.e. the *synthetic theory of evolution*, is given by Bäck (1996) and therefore omitted here. The more interested reader is referred to the book by Futuyma (1986).)

Although nature offers a variety of schemes one central idea of how these operators may be formalized can be extracted from nature as well as from several approaches in the context of EAs. Whereas the general working mechanism of both operators is very simple, the different achievements concerning distinct applications may vary. In order not to focus on a special construction a more abstract view of both operators is presented here (sufficing in the present context).

Imagine a genotype $\boldsymbol{x} = (x_1, \ldots, x_n) \in X$ consisting of genes $x_i \in G, i \in 1, \ldots, n$ (n corresponds to the actual genotype length) from a gene space G. The deletion operator del may than be formalized as a function transforming a given individual $\boldsymbol{a} = (\boldsymbol{x}, s) \in I$ by deleting a gene x_i. If $I = X \times A_s$ is the individual space, where A_s is the strategy parameter space, which depends on the application and the EA, del has the form

$$\text{del} : I \rightarrow I, \text{ with } \text{del}(\boldsymbol{a}) = \text{del}(x_1, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_n, s) = (x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n, s) = \boldsymbol{a}'.$$

In most cases an application-dependent probability $p_{\text{del}} \in (0, 1)$ is responsible for the decision whether deletion should be applied or not. The position i fixing the gene which has to be deleted is usually uniformly chosen from the set $\{1, \ldots, n\}$. Since deletion and duplication produce genotypes of different length it is important to notice that the dimension n varies from individual to individual. Returning to our example (Davidor 1991a), deletions occur only after a recombination ($p_c = 1.0$) and typically have a probability of 0.05. A deleted gene x_i has the form $x_i \in \mathbb{B}^l \times \mathbb{B}^l \times \mathbb{B}^l$, where each bit vector of length l codes for an angle.

Similar to deletion, the duplication operator is simple. Generally, instead of deletion, a gene is added to the genotype, such that the operator may be formalized as follows:

$$\text{dup} : I \to I, \text{ with } \text{dup}(\boldsymbol{a}) = \text{dup}(x_1, \ldots, x_i, \ldots, x_n, s) = (x_1, \ldots, x_i, x_i', \ldots, x_n, s) = \boldsymbol{a}'.$$

Analogously to deletion a duplication probability $p_{\text{dup}} \in (0, 1)$ is used and the index i is usually uniformly chosen. Concerning the policy for introducing the new gene x_i' several policies may be distinguished, such as:

- *Duplication.* The gene x_i' is a duplicate of x_i, such that $\boldsymbol{a}'$ has the form $\boldsymbol{a}' = (x_1, \ldots, x_i, x_i, \ldots, x_n, s)$.
- *Related.* The initialization of the new gene x_i' is context dependent: x_i' is generated with help the of the actual values of x_i and x_{i+1}.
- *Addition.* x_i' is initialized at random.

For example, Davidor (1991a) performs a duplication with a probability of 0.06 only when a recombination takes place. Whereas the duplication and addition policy is intuitive the related policy may be further divided into two strategies. First, 'the added arm-configuration is such that either its end-effector is positioned at the mid distance between the two adjacent end-effector positions, or its links have a mid metric value between the corresponding link positions in the adjacent arm-configurations'.

Finally, both operators have to adapt the length of the parameter vector $s \in A_s$. Because this process depends on the form of A_s details are omitted here.

C3.4.3.4 Problems arising when using variable-length genotypes

Despite the fact that variable-length genotypes may enhance the computational power of EAs (see motivations (ii) and (iii)), the introduction of this new concept borrowed from nature leads to several problems.

- The role of positions in a variable-length genotype is destroyed: the assignment of corresponding genes x_i on different 'homologous' chromosomes is not possible. In order to construct genetic operators which are able to generate interpretable individuals, thus being able to respect semantical blocks on the genotype, the 'assignment problem' has to be solved.
- In particular recombination is faced with the problem of finding the locus of corresponding genes.
- Whereas some authors introduced gene duplication and gene deletion operators 'in order to improve the stability of the strings' length' (Davidor 1991a, p 84) others waive these operators; that is, they believe that variable-dimensional recombination suffices for the stabilization of string lengths (see e.g. Harp and Samad 1991).

C3.4.3.5 Solutions

The *evolution program approach* of Michalewicz (1992), i.e. combining the concept of evolutionary computation with problem-specific chromosome structures and genetic operators, may be seen as one main concept used to overcome the problems mentioned above. Although this concept is useful in practice, it prevents the conception of a more general and formalized view of variable-length EAs because there no longer exists 'the EA' using 'the representation' and 'the set of operators'. Instead, for each application problem a specialized EA exists. According to Lohmann (1992) and Kost (1993), for example, the formulation of operators such as gene duplication and deletion, used in their framework of *structural evolution*, is strongly application dependent, thus inhibiting a more formal, general concept of these operators. Davidor (1991a, b) expressed the need for revised and new genetic operators for his variable-length robot trajectory optimization problem. In contrast to the evolution program approach, Schütz (1994) formulated an application-independent, variable-dimensional mixed-integer evolution strategy (ES), thus following the course of constructing a more general sort of ES. This offered Schütz the possibility to be more formal than other researchers. Unfortunately, this approach is restricted to a class of problems which can easily be mapped onto the mixed-integer representation he used.

Because most work concerning variable-length genotypes uses the evolution program approach, a formal analysis of gene duplication and deletion is rarely found in the literature and is therefore omitted here. As a consequence, theoretical knowledge about the behavior of gene duplication and deletion is nearly unknown. Harvey (1993), for example, points out 'that gene-duplication, followed by mutation of one of the copies, is potentially a powerful method for evolutionary progress'. Most statements concerning

nonstandard operators such as duplication and deletion have the same quality as Harvey's: they are far from being provable.

Because of the lack of theoretical knowledge we proceed by discussing some solutions used to circumvent the problems which arise when introducing variable-length genotypes. In the first place, we question how other researchers have solved the problem of noncomparable loci, i.e. the problem of respecting the semantics of loci. Mostly this 'gene assignment' problem is solved by explicitly marking semantical entities on the genotype. The form of the tagging varies from application to application and is carried out with the help of different representations.

- Davidor (1991a, b) used a binary encoded non-fixed-length vector of arm configurations, i.e. a vector of triples (three angles), for representing a robot trajectory, thus defining semantical blocks.
- Section G3.6 of this handbook discusses the path of a *mobile robot* as a variable-dimensional list of path nodes (triples consisting of the two Cartesian coordinates and a flag indicating whether a node is feasible or not). G3.6
- Harp and Samad (1991) implemented the tagging with the help of a special and more complex data structure representing the structure and actual weights of any feedforward net consisting of a variable number of hidden layers and a variable number of units.
- Goldberg *et al* (1989, 1990) extended the usual string representation of GAs by using a list of ordered pairs, with the first component of each tuple representing the position in the string and the second one denoting the actual bit value. Using genotypes of fixed length a variable dimension in the resulting *messy GA* was achieved by allowing strings not to contain full gene complement (underspecification) and redundant or even contradictionary genes (overspecification). C4.2.4
- Koza (1992, 1994) used rooted point-labeled trees with ordered branches (LISP expressions), thus having a genotype representing semantics very well.

Lohmann (1992) circumvented the 'assignment problem' using so-called *structural evolution*. The basic idea of structural evolution is the separation of structural and nonstructural parameters, thus leading to a 'two-level' ES: a multipopulation ES using isolation. While on the level of each population a parameter optimization, concerning a fixed structure, is carried out, on the population level several isolated structures compete with each other. In this way Lohmann was able to handle structural optimization problems with variable dimension: the dimension of the structural parameter space does not have to be constant. Since each ES itself worked on a fixed number of nonstructural parameters (here a vector of reals) no problem occurred on this level. On the structural level (population level) special genetic operators and a special selection criterion were formulated. The criticism concerning structural evolution definitively lies in the basic assumption that structural and nonstructural parameters can always be separated. Surely, many mixed-integer variable-dimensional problems are not separable. Secondly, on the structural level the well-known semantical problem exists, but was not discussed. Schütz (1994) totally omitted a discussion concerning semantical problems arising from variable-length genotypes.

If the genotype is sufficiently prepared, problems (especially) concerning recombination disappear, because the genetic operators may directly use the tagging in order to construct interpretable individuals. Another important idea when designing recombination operators for variable-length genotypes is pointed out by Davidor (1991a). He suggests a matching of parameters according to their genotypic character instead of to their genotypic position. Essentially, this leads to a matching on the phenotypic, instead of the genotypic level. Generally, Davidor points out:

> In a complex string structure where the number, size and position of the parameters has no rigid structure, it is important that the crossover occurs between sites that control the same, or at least the most similar, function in the phenotypic space.

In case of the (two-point) segregation crossover used in his robot trajectory optimization problem, crossing sites were specified according to the proximity of the end effector positions.

C3.4.3.6 Conclusions

One may remark that many ideas concerning the use of gene duplication and deletion exist. Unfortunately, most thoughts have been extremely application oriented, that is, not formulated generally enough. Probably the construction of a formal frame will be very complicated in the face of the diversity of problems and solutions.

References

Ackley D and Littman M 1991 Interactions between learning and evolution *Artificial Life II (Santa Fe, NM, February 1990)* ed C Langton, C Taylor, D Farmer and S Rasmussen (Redwood City, CA: Addison-Wesley) pp 487–509

——1994 A case for Lamarckian evolution *Artificial Life III* ed C Langton (Redwood City, CA: Addison-Wesley) pp 3–10

Anderson R W 1995a Learning and evolution: a quantitative genetics approach *J. Theor. Biol.* **175** 89–101

——1995b Genetic mechanisms underlying the Baldwin effect are evident in natural antibodies *Proc. 4th Ann. Conf. on Evolutionary Programming (San Diego, CA, March 1995)* ed J R McDonnell, R G Reynolds and D B Fogel (Cambridge, MA: MIT Press) pp 547–63

——1996a How adaptive antibodies facilitate the evolution of natural antibodies *Immunol. Cell Biology* **74** 286–91

——1996b Random-walk learning: a neurobiological correlate to trial-and-error *Prog. Neural Networks* at press

Bäck T 1996 *Evolutionary Algorithms in Theory and Practice* (New York: Oxford University Press)

Balakrishnan K and V Honavar 1995 *Evolutionary Design of Neural Architectures: a Preliminary Taxonomy and Guide to Literature* Artificial Intelligence Research Group, Department of Computer Science, Iowa State University, Technical Report CS TR 95-01

Baldwin J M 1896 A new factor in evolution *Am. Naturalist* **30** 441–51

Belew R K 1989 When both individuals and populations search: adding simple learning to the genetic algorithm *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, June 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 34–41

——1990 Evolution, learning and culture: computational metaphors for adaptive search *Complex Syst.* **4** 11–49

Bremermann H J and Anderson R W 1991 How the brain adjusts synapses—maybe *Automated Reasoning: Essays in Honor of Woody Bledsoe* ed R S Boyer (New York: Kluwer) pp 119–47

Bulmer M G 1985 *The Mathematical Theory of Quantitative Genetics* (Oxford: Oxford University Press) pp 150–2

Cecconi F, Menczer F and Belew R K 1995 *Maturation and the Evolution of Imitative Learning in Artificial Organisms* Technical Report CSE 506, University of California, San Diego; 1996 *Adaptive Behavior* **4** at press

Charlesworth B 1993 The evolution of sex and recombination in a varying environment *J. Heredity* **84** 345–50

Conrad M 1993 Structuring adaptive surfaces for effective evolution *Proc. 2nd Ann. Conf. on Evolutionary Programming (San Diego, CA)* ed D B Fogel and W Atmar (La Jolla, CA: Evolutionary Programming Society) pp 1–10

Conrad M and Ebeling W 1992 M V Volkenstein, evolutionary thinking and the structure of fitness landscapes *BioSystems* **27** 125–8

Davidor Y 1991a *Genetic Algorithms and Robotics. A Heuristic Strategy for Optimization (World Scientific Series in Robotics and Automated Systems 1)* (Singapore: World Scientific)

——1991b A genetic algorithm applied to robot trajectory generation *Handbook of Genetic Algorithms* ed L Davis (New York: Van Nostrand Reinhold) ch 12, pp 144–65

Davis L (ed) 1991 *Handbook of Genetic Algorithms* (New York: Van Nostrand Reinhold)

Fogel D B 1988 An evolutionary approach to the traveling salesman problem *Biological Cybern.* **60** 139–44

——1993 Applying evolutionary programming to selected traveling salesman problems *Cybern. Syst.* **24** 27–36

Fogel L J, Owens A J and Walsh M J 1966 *Artificial Intelligence through Simulated Evolution* (New York: Wiley)

Fontanari J F and Meir R 1990 The effect of learning on the evolution of asexual populations *Complex Syst.* **4** 401–14

French R and Messinger A 1994 Genes, phenes and the Baldwin effect *Artificial Life IV (Cambridge, MA, July 1994)* ed R A Brooks and P Maes (Cambridge, MA: MIT Press) pp 277–82

Futuyma D J 1986 *Evolutionary Biology* (Sunderland, MA: Sinauer)

Gehlhaar D K, Verkhivker G, Rejto P A, Fogel D B, Fogel L J and Freer S T 1995 Docking conformationally flexible small molecules into a protein binding site through evolutionary programming *Proc. 4th Ann. Conf. on Evolutionary Programming (San Diego, CA, March 1995)* ed J R McDonnell, R G Reynolds and D B Fogel (Cambridge, MA: MIT Press) pp 615–27

Goldberg D E, Deb K and Korb B 1990 Messy genetic algorithms revisited: studies in mixed size and scale *Complex Syst.* **4**

Goldberg D E, Korb B and Deb K 1989 Messy genetic algorithms: motivation, analysis, and first results *Complex Syst.* **3**

Grefenstette J J, Gopal R, Rosmaita B and Van Gucht D 1985 Genetic algorithms for the traveling salesman problem *Proc. Int. Conf. on Genetic Algorithms and their Applications* ed J J Grefenstette (Erlbaum) pp 160–6

Harp S A and Samad T 1991 Genetic synthesis of neural network architecture *Handbook of Genetic Algorithms* ed L Davis (New York: Van Nostrand Reinhold) ch 15, pp 202–21

Harvey I 1993 *The Artificial Evolution of Adaptive Behaviour* Master's Thesis, University of Sussex

Hightower R, Forrest S and Perelson A 1996 The Baldwin effect in the immune system: learning by somatic hypermutation *Adaptive Individuals in Evolving Populations: Models and Algorithms* ed R K Belew and M Mitchell (Reading, MA: Addison-Wesley) at press

Hinton G E and Nowlan S J 1987 How learning can guide evolution *Complex Syst.* **1** 495–502

Holland J H 1975 *Adaptation in Natural and Artificial Systems* (Ann Arbor, MI: University of Michigan Press)

Kost B 1993 *Structural Design via Evolution Strategies* Internal Report, Department of Bionics and Evolution Technique, Technical University of Berlin

Koza J R 1992 *Genetic Programming* (Cambridge, MA: MIT Press)

——1994 *Genetic Programming II* (Cambridge, MA: MIT Press)

Lin S and Kernighan B W 1976 An effective heuristic for the traveling salesman problem *Operat. Res.* **21** 498–516

Lohmann R 1992 Structure evolution and incomplete induction *Parallel Problem Solving from Nature, 2 (Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature, Brussels, 1992)* ed R Männer and B Manderick (Amsterdam: Elsevier) pp 175–85

Männer R and Manderick B (eds) 1992 *Parallel Problem Solving from Nature, 2 (Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature, Brussels, 1992)* ed R Männer and B Manderick (Amsterdam: Elsevier)

Maynard Smith J 1978 *The Evolution of Sex* (Cambridge: Cambridge University Press)

——1987 When learning guides evolution *Nature* **329** 761–2

Michalewicz Z 1992 *Genetic Algorithms + Data Structures = Evolution Programs* (Berlin: Springer)

Milstein C 1990 The Croonian lecture 1989 Antibodies: a paradigm for the biology of molecular recognition *Proc. R. Soc.* B **239** 1–16

Mitchell M and Belew R K 1995 Preface to G E Hinton and S J Nowlan How learning can guide evolution *Adaptive Individuals in Evolving Populations: Models and Algorithms* ed R K Belew and M Mitchell (Reading, MA: Addison-Wesley)

Morgan C L 1896 On modification and variation *Science* **4** 733–40

Osborn H F 1896 Ontogenic and phylogenic variation *Science* **4** 786–9

Nolfi S, Elman J L and Parisi D 1994 Learning and evolution in neural networks *Adaptive Behavior* **3** 5–28

Paechter B, Cumming A, Norman M and Luchian H 1995 Extensions to a memetic timetabling system *Proc. 1st Int. Conf. on the Practice and Theory of Automated Timetabling (ICPTAT 95) (Edinburgh, 1995)*

Parisi D, Nolfi S and Cecconi F 1991 Learning, behavior, and evolution *Toward a Practice of Autonomous Systems (Proc. 1st Eur. Conf. on Artificial Life (Paris, 1991))* ed F J Varela and P Bourgine (Cambridge, MA: MIT Press)

Rechenberg I 1973 *Evolutionsstrategie: Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution* (Stuttgart: Fromman-Holzboog)

Saravanan N, Fogel D B and Nelson K M 1995 A comparison of methods for self-adaptation in evolutionary algorithms *BioSystems* **36** 157–66

Scheiner S M 1993 Genetics and evolution of phenotypic plasticity *Ann. Rev. Ecol. Systemat.* **24** 35–68

Schütz M 1994 *Eine Evolutionsstrategie für gemischt-ganzzahlige Optimierungsprobleme mit variabler Dimension* Diploma Thesis, University of Dortmund

Schwefel H P 1968 *Projekt MHD-Staustrahlrohr: Experimentelle Optimierung einer Zweiphasendüse* Teil I Technischer Bericht 11.034/68, 35, AEG Forschungsinstitut, Berlin

Sober E 1994 The adaptive advantage of learning and *a priori* prejudice *From a Biological Point of View: Essays in Evolutionary Philosophy* (a collection of essays by E Sober) (Cambridge: Cambridge University Press) pp 50–70

Stearns S C 1989 The evolutionary significance of phenotypic plasticity—phenotypic sources of variation among organisms can be described by developmental switches and reaction norms *Bioscience* **39** 436–45

Stephens D W 1993 Learning and behavioral ecology: incomplete information and environmental predictability *Insect Learning. Ecological and Evolutionary Perspectives* ed D R Papaj and A C Lewis (New York: Chapman and Hall) ch 8, pp 195–218

Turney P D 1995 Cost-sensitive classification: empirical evaluation of a hybrid genetic decision tree induction algorithm *J. Artificial Intell. Res.* **2** 369–409

——1996 How to shift bias: lessons from the Baldwin effect *Evolutionary Comput.* at press

Turney P D, Whitley D and Anderson R W (eds) 1996 Special issue on evolution, learning, and instinct: 100 years of the Baldwin effect *Evolutionary Comput.* at press

Unemi T, Nagayoshi M, Hirayama N, Nade T, Yano K and Masujima Y 1994 Evolutionary differentiation of learning abilities—a case study on optimizing parameter values in Q-learning by a genetic algorithm *Artificial Life IV (July 1994)* ed R A Brooks and P Maes (Cambridge, MA: MIT Press) pp 331–6

Via S 1993 Adaptive phenotypic plasticity: target or by-product of selection in a variable environment? *Am. Naturalist* **142** 352–65

Waddington C H 1942 Canalization of development and the inheritance of acquired characters *Nature* **150** 563–5

Wcislo W T 1989 Behavioral environments and evolutionary change *Ann. Rev. Ecol. Systemat.* **20** 137–69

West-Eberhard M J 1989 Phenotypic plasticity and the origins of diversity *Ann. Rev. Ecol. Systemat.* **20** 249–78

Whitley D and Gruau F 1993 Adding learning to the cellular development of neural networks: evolution and the Baldwin effect *Evolutionary Comput.* **1** 213–33

Whitley D, Gordon S and Mathias K 1994 Lamarckian evolution, the Baldwin effect and function optimization *Parallel Problem Solving from Nature—PPSN III (Proc. Int. Conf. on Evolutionary Computation and 3rd Conf. on Parallel Problem Solving from Nature, Jerusalem, October 1994) (Lecture Notes in Computer Science 866)* ed Yu Davidor, H P Schwefel and R Männer (Berlin: Springer) pp 6–15

Wright S 1931 Evolution in Mendelian populations *Genetics* **16** 97–159

Further reading

More extensive treatments of issues related to the Baldwin effect can be found in the literature cited in section C3.4.1. The following are notable foundation and review papers.

1. Anderson R W 1995a Learning and evolution: a quantitative genetics approach *J. Theor. Biol.* **175** 89–101
2. Balakrishnan K and V Honavar 1995 *Evolutionary Design of Neural Architectures: a Preliminary Taxonomy and Guide to Literature* Artificial Intelligence Research Group, Department of Computer Science, Iowa State University, Technical Report CS TR 95-01
3. Baldwin J M 1896 A new factor in evolution *Am. Naturalist* **30** 441–51
4. Belew R K 1989 When both individuals and populations search: adding simple learning to the genetic algorithm *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, June 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 34–41
5. Hinton G E and Nowla Hinton G E and S J Nowlan 1987 How learning can guide evolution *Complex Syst.* **1** 495–502
6. Morgan C L 1896 On modification and variation *Science* **4** 733–40
7. Sober E 1994 The adaptive advantage of learning and *a priori* prejudice *From a Biological Point of View: Essays in Evolutionary Philosophy* (a collection of essays by E Sober) (Cambridge: Cambridge University Press) pp 50–70
8. Turney P D 1995 Cost-sensitive classification: empirical evaluation of a hybrid genetic decision tree induction algorithm *J. Artificial Intell. Res.* **2** 369–409
9. Turney P D, Whitley D and Anderson R W (eds) 1996 Special Issue on evolution, learning, and instinct: 100 years of the Baldwin effect *Evolutionary Comput.* at press
10. Waddington C H 1942 Canalization of development and the inheritance of acquired characters *Nature* **150** 563–5
11. Wcislo W T 1989 Behavioral environments and evolutionary change *Ann. Rev. Ecol. Systemat.* **20** 137–69
12. Whitley D and F Gruau 1993 Adding learning to the cellular development of neural networks: evolution and the Baldwin effect *Evolutionary Comput.* **1** 213–33

C4.1 Introduction

Hitoshi Iba

Abstract

This section introduces the fitness evaluation for evolutionary algorithms and briefly describes the related problems discussed in the rest of this chapter.

C4.1.1 Fitness evaluation

First, we describe how to encode and decode for fitness evaluations. Most *genetic algorithms* require encoding; that is, the mapping from the chromosome representation to the domain structures (e.g. parameters). The recombination operators (i.e. crossover or mutation) work directly on this coded representation, not on the domain structures. More formally, suppose that an optimization problem is given as follows: B1.2

$$f : M \longrightarrow \Re \tag{C4.1.1}$$

where M is the search space of the objective function f. Then the fitness evaluation function F is described as follows:

$$F : R \xrightarrow{d} M \xrightarrow{f} \Re \xrightarrow{s} \Re_+ \tag{C4.1.2}$$

$$F = s \circ f \circ d \tag{C4.1.3}$$

where R is the space of the chromosome representation, d is a decoding function, and s is a scaling function. The scaling function s is typically used in combination with *proportional selection* in order to guarantee positive fitness values and fitness maximization. For instance, when encoding an n-dimensional real-valued objective function f_n by binary coding, the above fitness function F is given as follows: C2.2

$$F : \{0, 1\}^l \xrightarrow{d_\mathrm{b}} \Re^n \xrightarrow{f_n} \Re \xrightarrow{s} \Re_+ \tag{C4.1.4}$$

where l is the length of a chromosome and d_b is the binary coding; that is, d_b maps segments of the chromosome into real numbers of corresponding dimensions. The evaluations of the chromosomes are converted into fitness values in various ways. For instance, there are many coding schemes using a binary character set that can code a parameter with the same meaning, such as a binary code and a Gray code. However, experimental results have shown that Gray coding is superior to binary coding for a particular function optimization for genetic algorithms (GAs) (see e.g. Caruana and Schaffer 1988). Analysis suggested that Gray coding eliminates the 'Hamming cliff' problem that makes some transitions difficult for a binary representation (see Bethke 1981, Caruana and Schaffer 1988, and Bäck 1993 for details). Therefore, the encoding scheme is very important to improve the search efficiency of GAs. The details are discussed in Section C4.2. In contrast to GAs, *evolution strategies* (ESs) and *evolutionary programming* work directly on the second space M, such that they do not require the decoding function d. Furthermore, they typically do not need a scaling function s, such that the fitness evaluation function is fully specified by equation (C4.1.1). C4.2, B1.3, B1.4

C4.1.2 Related problems

Second, it is often difficult to compute a solution with global accuracy for complex problems. The difficulty stems from the objectivity of the fitness function, which often comes only at the cost of significant knowledge about the search space (Angeline 1993). In order to eliminate the reliance on objective fitness functions, a competition is introduced. The competitive fitness function is a method for calculating fitness that is dependent on the current population, whereas the standard fitness functions return the same fitness for an individual regardless of other members in the population. The advantage of the competition is that evolutionary algorithms do not need an exact fitness value (i.e. the above f value), because most selection schemes work by just comparing fitness; that is the 'better or worse' criterion suffices. In other words, the absolute measure of fitness value is not required, but the relative measure when the individual is tested against other individuals should be derived. Thus, this method is computationally more efficient and more amenable to parallel implementation. The details are described in Section C4.3. C4.3

The third problem arises from the difficulty of evaluating the tradeoff between the fitness of a genotype and its complexity. For instance, the fitness definitions used in traditional genetic programming do not include evaluations of the tree descriptions. Therefore without the necessary control mechanisms, trees may grow exponentially large or become so small that they degrade search efficiency. Usually the maximum depth of trees is set as a user-defined parameter in order to control tree sizes, but an appropriate depth is not always known beforehand. For this purpose, we describe in Section C4.4 the complexity-based fitness evaluation by employing statistical measures, such as AIC and MDL. C4.4

Many real-world problems require a simultaneous optimization of multiple objectives. It is not necessarily easy to search for the different goals especially when they conflict with each other. Thus, there is a need for techniques different from the standard optimization in order to solve the multiobjective problem. Section C4.5 introduces several GA techniques studied recently. C4.5

References

Angeline P J 1993 *Evolutionary Algorithms and Emergent Intelligence* Doctoral Dissertation, Ohio State University

Bäck T 1993 Optimal mutation rates in genetic search *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 2–9

Bethke A D 1981 *Genetic Algorithms as Function Optimizers* Doctoral Dissertation, University of Michigan

Caruana R A and Schaffer J D 1988 Representation and hidden bias: Gray vs binary coding for genetic algorithms *Proc. 5th Int. Conf. on Machine Learning (Ann Arbor, MI, 1988)* ed J Laird (San Mateo, CA: Morgan Kaufmann) pp 153–61

C4.2 Encoding and decoding functions

Kalyanmoy Deb

Abstract

Encoding and decoding functions are relavent to the studies of genetic algorithms (GAs), because GAs work with a coding of variables. A number of encoding and decoding schemes have been used in GA studies for this purpose. An encoding function is used to code the object variables in a string structure. In order to retrieve the object variables from a string, a decoding function are used. Although in most studies, the binary coding has been used, the Gray coding is also becoming popular in the recent past. Besides binary and Gray coding, we also discuss some of the other coding schemes such as messy coding and floating-point coding, and briefly describe coding schemes used in solving permutation and control system problems.

C4.2.1 Introduction

Among the EC algorithms, *genetic algorithms* (GAs) work with a coding of the object variables, instead of the variables directly. Thus, the encoding and decoding schemes are more relevant in the studies of GAs. *Evolution strategy* (ES) and *evolutionary programming* (EP) methods directly use the object variables. Thus, no coding is used in these methods. *Genetic programming* (GP) uses LISP codes to represent a task and no special coding scheme is usually used. B1.2 B1.3, B1.4 B1.5.1

GAs begin their search by creating a population of solutions which are represented by a coding of the object variables. Before the fitness of each solution can be calculated, each solution must be decoded to obtain the object variables with a decoding function, $\gamma : \mathbb{B} \rightarrow \mathbb{M}$, where $\mathbb{M}$ represents a problem-specific space. Thus, the decoding functions are more useful from the GA implementation point of view, whereas the encoding functions (γ^{-1}) are important for understanding the coding aspects. The objective function ($f : \mathbb{M} \rightarrow \mathbb{R}$) in a problem can be calculated from the object variables defined in the problem-specific space $\mathbb{M}$. Thus the fitness function is a transformation defined as follows: $\Phi = f \circ \gamma$. It is important to mention here that both the above functions play an important role in the working of GAs. In addition, in the calculation of the fitness function, a scaling function is sometimes used after the calculation of f and γ functions. The scaling functions are described in Section C2.2. In the following subsections, we discuss different encoding and decoding schemes used in GA studies. C2.2

C4.2.2 Binary strings

In most applications of GAs, *binary strings* are used to encode object variables. A binary string is defined using a binary alphabet $\{0, 1\}$. An l-bit string occupies a space $\mathbb{B}^l = \{0, 1\}^l$. Each object variable is encoded in a binary string of a particular length l_i defined by the user. Thereafter, a complete l-bit string is formed by concatenating all substrings together. Thus, the complete GA string has a length l: C1.2

$$l = \sum_{i=1}^{n} l_i \tag{C4.2.1}$$

where n is the number of object variables. A binary string of length l_i has a total of 2^{l_i} search points. The string length used to encode a particular variable depends on the desired precision in that variable.

A variable requiring a larger precision needs to be coded with a larger string and vice versa. A typical encoding of n object variables $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$ into a binary string is illustrated in the following:

$$\overbrace{\underbrace{100\ldots1}_{l_1}}^{x_1} \quad \overbrace{\underbrace{010\ldots0}_{l_2}}^{x_2} \quad \cdots \quad \overbrace{\underbrace{110\ldots0}_{l_n}}^{x_n}.$$

The variable x_1 has a string length l_1 and so on. This encoding of the object variables to a binary string allows GAs to be applied to a wide variety of problems. This is because GAs work with the string and not with the object variables directly. The actual number of variables and the range of search domain of the variables used in the problem are masked by the coding. This allows the same GA code to be used in different problems without much change.

The decoding scheme used to extract the object variables from a complete string works in two steps. First, the substring $(a_{i1}, \ldots, a_{il_i})$, where $a_{ij} \in \mathbb{B}^{l_i}$, corresponding to each object variable is extracted from the complete string. Knowing the length of the substring and lower (u_i) and upper bounds (v_i) of the object variables, the following linear decoding function is mostly used $(\gamma_i : \mathbb{B}^{l_i} \rightarrow [u_i, v_i])$:

$$x_i = u_i + \frac{v_i - u_i}{2^{l_i} - 1} \left(\sum_{j=0}^{l_i - 1} a_{i(l_i - j)} 2^j \right). \tag{C4.2.2}$$

The above decoding function linearly maps the decoded integer value of the binary substring in the desired interval $[u_i, v_i]$. The above operation is carried out for all object variables. Thus, the operation $\gamma = \gamma_1 \times \ldots \times \gamma_n$ yields a vector of real values by interpreting the bit string as a concatenation of binary encoded integers mapped to the desired real space. As seen from the above decoding function, the maximum attainable precision in the ith object variable is $(v_i - u_i)/(2^{l_i} - 1)$. Knowing the desired precision and lower and upper bounds of each variable, a lower bound of the string size required to code the variable can be obtained.

Although the binary strings have been mostly used to encode object variables, higher-ary alphabets have also been used in some studies. In those cases, instead of a binary alphabet a higher-ary alphabet is used in a string. For a χ-ary alphabet string of length l, there are a total of χ^l strings possible. Although the search space is larger with a higher-ary alphabet coding than with a binary coding of the same length, Goldberg (1990) has shown that the schema processing is maximum with binary alphabets. Nevertheless, in χ-ary alphabet coding the decoding function in equation (C4.2.2) can be used by replacing 2 with χ.

In the above binary string decoding, the object variables are assumed to have a uniform search interval. For nonuniform but defined search intervals (such as exponentially distributed intervals and others), the above decoding function can be suitably modified. However, in some real-world search and optimization problems, the allowable values of the object variables do not usually follow any pattern. In such cases, the binary coding can be used, but the corresponding decoding function becomes cumbersome. In such cases, a look-up table relating a string and the corresponding value of the object variable is usually used (Deb and Goyal 1996).

Often in search and optimization problems, some object variables are allowed to take both negative and positive values. If the search interval in those variables is the same in negative and positive real space $(x_i \in \{-u_i, u_i\})$, a special encoding scheme is sometimes used. The first bit can be used to encode the sign of the variables and the rest $(l_i - 1)$ bits can be used to encode the magnitude of the variable (searching in the range $\{0, u_i\}$). It turns out that this encoding scheme is not very different from the simple binary encoding scheme applied over the entire search space.

C4.2.3 Gray coded strings

Often, a *Gray coding* with binary alphabets is used to encode object variables (Caruna and Schaffer 1988, Schaffer *et al* 1989). Like the binary string, a Gray-coded string is also collection of binary alphabets of 1s and 0s. But the encoding and decoding schemes to obtain object variables from Gray-coded strings and vice versa are different. The encoding of the object variables to a Gray-coded string works in two steps. From the object variables, a corresponding binary string needs to be created. Thereafter, the binary string can be converted into a corresponding Gray code. A binary string $(b_1, b_2, \ldots, b_l)$, where $b_i \in \{0, 1\}$, is

converted to a Gray code $(a_1, a_2, \ldots, a_l)$ by using a mapping $\gamma^{-1} : \mathbb{B}^l \rightarrow \mathbb{B}^l$ (Bäck 1993):

$$a_i = \begin{cases} b_i & \text{if } i = 1 \\ b_{i-1} \oplus b_i & \text{otherwise} \end{cases} \tag{C4.2.3}$$

where $\oplus$ denotes addition modulo 2. As many researchers have indicated, the main advantage of a Gray code is its representation of adjacent integers by binary strings of Hamming distance one.

The decoding of a Gray-coded string into the corresponding object variables also works in two steps. First, the Gray-coded string $(a_1, \ldots, a_l)$ is converted into a simple binary string $(b_1, \ldots, b_l)$ as follows:

$$b_i = \bigoplus_{j=1}^{i} a_j \qquad \text{for } i = \{1, \ldots, l\}. \tag{C4.2.4}$$

Thereafter, a decoding function similar to the one described in section C4.2.2 can be used to decode the binary string into a real number in the desired range $[u_i, v_i]$.

C4.2.4 Messy coding

In the above coding schemes, the position of each gene is fixed along the string and only the corresponding bit value is specified. For example, in the binary string (`101`), the first and third genes take a value `1` and the second gene takes the value `0`. If in a problem, a particular bit combination for some widely separated genes constitute a building block, it will be difficult to maintain the building block in the population under the action of the recombination operator. This problem is largely known as the *linkage* problem in GAs (Although a natural choice to bring the right gene combinations together is to use an *inversion* operator, Goldberg and Lingle (1985) have argued that inversion does not have an adequate search power to do the task in a reasonable time.) In order to solve the linkage problem, a different encoding scheme is suggested by Goldberg *et al* (1989). Both the gene position and the corresponding bit values are coded in a string. A typical four-bit string is coded as follows:

C3.1

```
((2 1) (4 0) (1 1) (3 1))
```

The first entry inside a parenthesis is the gene location and the second entry is the bit value for that position. In the above string, the second, first and third genes have a value `1` and the fourth gene has a value `0`. Thus, the above string is a representation of the binary string `1110`. Since the gene location is also coded, good and important gene combinations can be expressed *tightly* (that is, adjacent to each other). This will reduce the chance of disruption of important building blocks due to the recombination operator. For example, if the bit-combination of `1` at the first gene and `0` at the fourth gene constitute a building block to the underlying problem, the above string codes the building block adjacent to each other. Thus, it will have a lesser chance of disruption due to the action of the recombination operator. This encoding scheme has been used to solve deceptive problems of various complexity (Goldberg *et al* 1989, Goldberg *et al* 1990).

C4.2.5 Floating-point coding

Inspired by the success of the above flexible encoding scheme, Deb (1991) with assistance from Goldberg, has developed a floating-point encoding scheme for continuous variables. In that scheme, both mantissa and exponent of a floating-point parameter are represented by separate genes designated by `M` and `E`, respectively. For a multiparameter optimization problem, a typical gene has three elements, as opposed to two in the above encoding. The three elements are the parameter identification number, mantissa or exponent declaration, and its value. A typical two-parameter string is shown in the following:

```
((1 E +) (1 M 1) (1 E -) (2 M 1) (1 M 0) (1 M +) (2 E -) (2 E -) (1 M 0))
```

The decoding is achieved by first extracting mantissa and exponent values of each variable and then the parameter value is calculated using the following decoding function:

$$x_i = \text{mantissa}_i \times \text{base}^{\text{exponent}_i} \tag{C4.2.5}$$

where base is a fixed number (a value of 10 is suggested). A + and a − in the exponent gene indicate $+1$ and -1, respectively. In decoding the exponent of a parameter, first the number of + and − genes are counted. The exponent is then calculated by algebraically summing the number of + and − in the exponent genes. In the above string, the exponent in the first variable has one + and one −. Thus, the net exponent value is zero. For the second variable, there are no + and two −. Thus, the exponent value is -2. In order to decode the mantissa part of each variable, sets of 1 and 0 separated by either a + or a − are first identified in a left-to-right scan. Each set is then decoded by adaptively reducing interval depending on the length of each set. In the above string, the first variable has mantissa elements (in a left-to-right scan) 10+0. There are two sets of 1s and 0s that are separated by a +. In the first set, there are two bits with one 1 and one 0. With two bits, there are a total of three unary combinations possible: no 1, one 1, and two 1s. Dividing the mantissa search interval (0,1) into three intervals and denoting the first interval (0, 0.333) by no 1, the second interval (0.333, 0.667) by one 1, and the third interval (0.667, 1) by two 1s, we observe that the specified interval is the second interval. A + indicates that the decoded value of the next set has to be added to the lower limit of the current interval. Since in the next set there is only one bit, the current interval (0.333, 0.667) is now divided into two equal subintervals. Since the bit is a 0, we are in the first subinterval (0.333, 0.500). The decoded value of the mantissa of the first variable can then be taken as the average of the two final limits. As more mantissa bits are added to the right of the above string, the corresponding interval gets continuously reduced and the accuracy of the solution is improved. Thus, the decoded value of the first parameter is $0.416(10^0)$ or 0.416, and that of the second parameter is $0.75(10^{-2})$ or 0.0075. This flexibility in coding allows important mantissa-exponent combinations to be coded tightly, thereby reducing the chance of disruption of good building blocks. Deb (1991) has used this encoding-decoding scheme to solve a difficult optimization problem and an engineering design problem.

C4.2.6 Coding for binary variables

In some search problems, some of the object variables are binary denoting the presence or absence of a member. In network design problems, the presence of absence of a link in the network is often a object variable. In truss-structure design problems such as bridges and roofs, the presence or absence of a member is a design variable. In a neural network design problem, the presence or absence of a connection between two neurons is a decision variable. In these problems, the use of binary alphabets (1 for presence and 0 for absence) is most appropriate.

C4.2.7 Coding for permutation problems

In permutation problems, such as the *traveling salesperson problem* and *scheduling* and *planning* problems, G9.5, F1.5
usually a series of node numbers is used to encode a permutation (Starkweather *et al* 1991). Usually in such problems a valid permutation requires each node number to appear once and only once. In these problems, the relative positioning of the node numbers are more important than the absolute positioning of the node numbers (Goldberg 1989). Although the sequence of node numbers to represent a permutation makes the encoding and decoding simpler, a few other problem-specific coding schemes have also been used in permutation problems (Whitley 1989).

C4.2.8 Coding for control problems

In an *optimal control* problem, the decision variable is a time- or frequency-dependent function of some F1.3
control variables. In applications of EC methods to optimal control problems, the practice has been to discretize the total time or frequency domain into several intervals and use the value of the control parameter at the beginning of each interval as a vector of object variables. In the case of a time-dependent control function $c(t)$ (from $t = t_1$ to $t = t_n$), the object variable vector ($\boldsymbol{x} = \{x_1, \ldots . x_n\}$) is defined as follows ($\gamma^{-1} : \mathbb{R} \rightarrow \mathbb{R}$):

$$x_i = c(t = t_i). \tag{C4.2.6}$$

In order to decode a vector of object variables into a time- or frequency-dependent control function, piecewise spline approximation functions with adjacent object variables can be used (Goldberg 1989). In many optimal control problems, the control variable either monotonically increases or monotonically decreases with respect to the state variable (time of frequency). In such cases, an efficient coding scheme would be to use the difference in the control parameter values in two adjacent states, instead of the absolute value,

as the object variable. For example, in the case of monotonically-increasing control variable, the following object variables can be used ($\gamma^{-1} : \mathbb{R} \to \mathbb{R}$):

$$x_i = \begin{cases} c(t_1) & i = 1 \\ c(t_i) - c(t_{i-1}) & \text{otherwise.} \end{cases} \tag{C4.2.7}$$

The decoding function in this representation is little different than before. The control function can be formed by spline-fitting the adjacent control parameter values, which are calculated as follows ($\gamma : \mathbb{R} \to \mathbb{R}$):

$$c_i = \begin{cases} x_1 & i = 1 \\ x_i - x_{i-1} & \text{otherwise.} \end{cases} \tag{C4.2.8}$$

C4.2.9 Conclusions

Genetic algorithms, among other evolutionary computation methods, work mostly with a coding of object variables. The mapping of the object variables to a string code is achieved through an *encoding* function and the mapping of a string code to its corresponding object variable is achieved through a *decoding* function. In a binary coding, each object variable is discretized to take a finite number of values in a specified range. Although the binary coding has been popular, other coding schemes are also described, such as Gray coding to achieve a better variational property between encodings and corresponding decoded values, messy coding to achieve tight linkage of important gene combinations, and floating-point coding to have a unary coding scheme of real numbers. To take care of problems having binary object variables, permutation problems and optimal control problems using evolutionary computation algorithms, three different coding schemes are also discussed.

References

Bäck T 1993 Optimal mutation rates in genetic search *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 2–8

Caruana R A and Schaffer J D 1988 Representation and hidden bias: Gray versus binary coding in genetic algorithms *Proc. 5th Int. Conf. on Machine Learning (Ann Arbor, MI, 1988)* ed J Laird (San Mateo, CA: Morgan Kaufmann) pp 153–61

Deb K 1991 Binary and floating-point function optimization using messy genetic algorithms. (Doctoral dissertation, University of Alabama and IlliGAL Report No. 91004) *Dissertation Abstracts International* **52**(5), 2658B

Deb K and Goyal M 1996 A robust optimization procedure for mechanical component design based on genetic adaptive search *Technical Report No. IITK/ME/SMD-96001* Department of Mechanical Engineering, Indian Institute of Technology, Kanpur

Goldberg D E 1989 *Genetic Algorithms in Search, Optimization, and Machine Learning* (Reading, MA.: Addison-Wesley)

Goldberg D E 1990 Real-coded genetic algorithms, virtual alphabets, and blocking *IlliGal Report No 90001* (Urbana, IL: University of Illinois at Urbana-Champaign)

Goldberg D E, Deb K and Korb B 1990 Messy genetic algorithms revisited: Nonuniform size and scale *Complex Syst.* **4** 415–44

Goldberg D E, Korb B and Deb K 1989 Messy genetic algorithms: Motivation, analysis, and first results *Complex Syst.* **3** 493–530

Goldberg D E and Lingle R 1985 Alleles, loci, and the traveling salesman problem *Proc. 1st Int. Conf. on Genetic Algorithms (Pittsburgh, PA, July 1985)* ed J J Grefenstette (Hillsdale, NJ: Lawrence Erlbaum Associates) pp 154–9

Schaffer J D, Caruana R A, Eshelman L J, and Das R 1989 A study if control parameters affecting online performance of genetic algorithms for function optimization *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, June 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 51–60

Starkweather T, McDaniel S, Mathias K, Whitley D, and Whitley C 1991 A comparison of genetic sequencing operators *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, July 1991)* ed R Belew and L Booker (San Mateo, CA: Morgan Kaufmann) pp 69–76

Whitley D, Starkweather T and Fuquay D 1989 Scheduling problems and traveling salesman: The genetic edge recombination operator *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, June 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 133–40

C4.3 Competitive fitness evaluation

Peter J Angeline

Abstract

Competitive fitness evaluation is an alternative to standard 'objective' fitness evaluations that evaluate the worth of a population member through competition. This section describes the advantages of competitive fitness evaluation and compares several methods on the number of comparisons performed.

C4.3.1 Objective fitness

Typically in evolutionary computations, the value returned by the fitness function is considered to be an exact, objective measure of the absolute worth of the evaluated solution. More formally, the fitness function represents a complete ordering of all possible solutions and returns a value for a given solution that is related to its rank in the complete ordering. While in some environments such 'absolute objective' knowledge is easily obtained, it is often the case in real-world environments that such information is inaccessible or unrepresentable. Such fitness functions are sometimes called *objective fitness functions* (Angeline and Pollack 1993).

Consider the following situation: given solutions A, B, and C, assume that A is preferred to B, B is preferred to C, and C is preferred to A. Such a situation occurs often in games where the C strategy can beat the A strategy, but is beaten by the B strategy, which is in turn beaten by the A strategy. Note that an objective fitness function cannot accurately represent such an arrangement. If the objective fitness function gives A, B, and C the same fitness, then if only A and B are in the population, the fact that A should be preferred to B is unrepresented. The same holds for the other potential pairings. Similarly, if the fitness function assigns distinct values for the worth of A, B, and C, there will always be a pair of strategies that will have fitness values that do not reflect their actual worth. The actual worth of any of the strategies A, B, or C, in this case, is relative to the contents of the population. If all three are present then none is to be preferred over the others; however, if only two are present then there is a clear winner. In such problems, which are more prevalent than not, no objective fitness function can adequately represent the space of solutions and subsequently evolutionary computations will be misled.

C4.3.2 Relative and competitive fitness

Relative fitness measures are an alternative to objective measures. Relative fitness measures access a solution's worth through direct comparison to some other solution either evolved or provided as a component of the environment. Competitive fitness is one type of relative fitness measure that is sensitive to the contents of the population. Sensitivity to the population is achieved through direct competition between population members. For such fitness measures, an objective fitness function that provides a partial ordering of the possible solutions is not required. All that is required is a relative measure of 'better' to determine which of two competing solutions is preferred.

The chief advantage of a competitive fitness functions are that they are self-scaling. Early in the run, when the evolving solutions perform poorly on the task, a solution need not be proficient in order to survive and reproduce. As the run continues and the average ability of the population increases, the average level of proficiency of a surviving solution will be suitably higher. As the population becomes increasingly

better at solving the task, the difficulty of the fitness function is continually scaled commensurate with the ability of the average population members. Angeline and Pollack (1993) argue that competitive fitness measures set up an environment where complex ecologies of problem solving form that naturally encourage the emergence of generalized solutions.

Three types of competitive fitness function have been used in previous studies: full competition; bipartite competition; and a tournament competition. All of these are successful competitive fitness measures but they differ on the number of competitions required and necessity for additional objective measures.

Axelrod (1987) used a full competition where each player played every other player in the population, as is standard practice in game theory. The number of competitions required in such a scheme for a population of size N is

$$\frac{N(N-1)}{2}.$$

This is a considerable number of fitness evaluations, but it does provide a significant amount of information, and the number of competitions won by a player provides a sufficient amount of information for ranking the population members.

Hillis (1991) describes a genetic algorithm for evolving sorting networks using a bipartite competition scheme. In a bipartite competition, there are two teams (populations) and individuals from one team are played against individuals from the other. The total number of competitions played between population members is $N/2$ where N is the combined population size. While this method does provide significant feedback, it does not automatically produce a hierarchical ordering for the population. Consequently, an objective measure must be used to rank order the individuals at the completion of the competition for each of the two populations to determine which population members will reproduce.

Angeline and Pollack (1993) describe a single elimination tournament competitive fitness measure. In this method, each player is paired randomly with another player in the population with winners advancing to the next level of competition. Then all of the winners are again randomly paired and compete, with these winners again advancing to the next round. Play continues until a single individual is left, having beaten every competitor met in the tournament. This individual is designated as the best-of-run individual. The rank of the other population members is determined by the number of competitions they won before being eliminated. The number of competitions between N players in a single elimination tournament is in total $N-1$, which is one fewer than the number of competitions held if each player played a user-designated teacher. Angeline and Pollack (1993) illustrate the presence of noise inherent in this method of fitness computation but claim that it may actually promote a more diverse population and ultimately help the evolutionary process.

The drawback of this fitness method is that it is often not clear how to create competitive fitness functions for problems that are not inherently competitive.

References

Angeline P J and Pollack J B 1993 Competitive environments evolve better solutions for complex tasks *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 264–70

Axelrod R 1987 Evolution of strategies in the iterated prisoner's dilemma *Genetic Algorithms and Simulated Annealing* ed L Davis (Boston, MA: Pitman) pp 32–41

Hillis W D 1991 Co-evolving parasites improve simulated evolution as an optimization procedure *Emergent Computation* ed S Forrest (Cambridge, MA: MIT Press) pp 228–35

C4.4 Complexity-based fitness evaluation

Hitoshi Iba

Abstract

This section describes the complexity-based fitness evaluation for evolutionary algorithms. We first introduce and compare the leading competing model selection criteria, namely, an MDL (minimum-description-length) principle, the AIC (Akaike information criterion), an MML (minimum-message-length) principle, the PLS (predictive least-squares) measure, cross-validation, and the maximum-entropy principle. Then we give an illustrative example to show the effectiveness of the complexity-based fitness by experimenting with evolving decision trees using genetic programming (GP). Thereafter, we describe various research on complexity-based fitness evaluation, that is, controlling genetic algorithm or GP search strategies by means of the MDL criterion.

C4.4.1 Introduction

Complexity-based fitness is grounded on a *simplicity criterion*, which is defined as a limitation on the complexity of the model class that may be instantiated when estimating a particular function. For example, when one is performing a polynomial fit, it seems fairly apparent that the degree of the polynomial must be less than the number of data points. Simplicity criteria have been studied by statisticians for many years.

This section outlines and compares the leading competing model selection criteria, namely, an MDL (minimum-description-length) principle, the AIC (Akaike information criterion), an MML (minimum-message-length) principle, the PLS (predictive least-squares) measure, cross-validation, and the maximum-entropy principle.

C4.4.2 Model selection criteria

The complexity of an algorithm can be measured by the length of its minimal description in some language. The old but vague intuition of Occam's razor can be formulated as the *minimum-description-length criterion*; that is, given some data, the most probable model is the model that minimizes the sum (Weigend *et al* 1994):

$$\text{MDL(model)} = \text{description\_length(data given model)} + \text{description\_length(model)} \longrightarrow \min \quad \text{(C4.4.1)}$$

where description_length(data given model) is the code length of the data when encoded using the model as a predictor for the data. The sum MDL(model) represents the tradeoff between residual error (i.e. the first term) and model complexity (i.e. the second term) including a structure estimation term for the final model. The final model (with the minimal MDL) is optimum in the sense of being a consistent estimate of the number of parameters while achieving the minimum error (Tenorio and Lee 1990).

More formally, suppose that z_i is a sequence of observations from the random variable Z, which is characterized by probability function $p_Z(\theta)$. The dominant form of the MDL is

$$\text{MDL}(k) = -\log_2 p(z \mid \hat{\theta}) + \frac{k}{2} \log_2 N \quad \text{(C4.4.2)}$$

where $\hat{\theta}$ is the maximum-likelihood estimate of θ, $p(z \mid \hat{\theta})$ is the likelihood of the estimated density function of $p_Z(\theta)$, k is the number of parameters in the model and N is the number of observations. The first term is the self-information of the model, which can be interpreted as the number of bits necessary to encode the observations. The second term can be also interpreted as the number of bits needed to encode the parameters of the model. Hence, the model which achieves the minimum of MDL is the most efficient model to encode the observations (Tenorio and Lee 1990, p 103).

Another criterion is the *Akaike information criterion* (AIC) (Akaike 1977). The essential idea here is to establish how many parameters, k, to include in a model. Minimizing AIC means minimizing k minus the log-likelihood function for the model, based on some assumed variance, $\hat{\sigma}^2$. In particular, if k is allowed to become too large, it does not matter that the likelihood of the data given a k-parameter model is very great; one will not achieve a minimal AIC. Unfortunately, the log-likelihood function cannot be calculated without an assumed family of distributions and a reasonable estimate of $\hat{\sigma}^2$. Nevertheless, the AIC has an important structural feature and that is the existence of a penalty term for the model complexity (Seshu 1994, p 220). The AIC is an approximation of the idealized Kullback–Leibler distance between the 'true' data generating distribution and the model, which involves the expectation operation.

Assuming the above condition of $\{z_1, \ldots, z_N\}$, the AIC estimator is given as

$$\mathrm{AIC}(k) = -2 \ln p(z \mid \hat{\theta}) + 2k. \tag{C4.4.3}$$

By comparison with the MDL(k) criterion, we see that the difference is the crucial second term, k (AIC) versus $(k/2)\log_2 N$ (MDL). Therefore, if N is sufficiently large, the second term in equation C4.4.2 tends to penalize k much more severely for MDL than for AIC. The MDL(k) criterion penalizes the number of parameters asymptotically much more severely (Rissanen 1989, p 94). Moreover, under some conditions, it is assumed that learning generally converges much faster for MDL than for AIC (see the article by Yamanishi (1992) for details).

Wallace proposed a similar measure called MML (minimum message length). The coded form has two parts. The first states the inferred estimates of the unknown parameters in the model, and the second states the data using an optimal code based on the data probability distribution implied by these parameter estimates (Wallace and Freedman 1987). The total length might be interpreted as minus the log joint probability of estimate and data, and minimizing the length is therefore closely similar to maximizing the posterior probability of the estimate. MML is almost identical to MDL. However, they differ in the implementations and philosophical views as to prior probabilities. The details of these differences can be found in the article by Wallace and Freedman (1987) and its discussions.

The other criteria proposed are the *cross-validation* measure and the maximum-entropy principle. It is shown that qualitatively and asymptotically the cross-validation criterion is equivalent to AIC. We may consider that the maximum-entropy principle is a special case of the MDL principle, namely one where the model class is restricted to be of a special form. Within the statistical community, there is a considerable debate about both the proper viewpoint and the nature of the penalty term (Seshu 1994).

The goal shared by these complexity-based principles is to obtain accurate and parsimonious estimates of the probability distribution. The idea is to estimate the simplest density that has high likelihood by minimizing the total length of the description of the data. Barron introduced the index of resolvability, which may be interpreted as the MDL principle applied on the average. It is has been shown that the rate of convergence of minimum complexity estimators is bounded by the index of resolvability (Barron 1991).

Another useful criterion proposed is PLS (i.e. predictive least squares) or PSE (i.e. predicted square error) (Rissanen 1989, p 122). This is mostly aimed at solving the 'selection-of-variables' problem for linear regressions. The problem is solved by using the stochastic complexity and the sum of the prediction errors as a criterion, the latter either considered as an approximation of the former or as providing an independent extension of the LS principle. Rissanen described how to achieve the PLS solution to the posed regression problem, and revealed that the PLS criterion is a special case of the MDL principle. The detailed discussions are given by Rissanen (1989, chapter 5).

C4.4.3 An example: minimum-description-length-based fitness evaluation for genetic programming

As an illustrative example, we present results of the experiments to evolve decision trees for Boolean concept learning using *genetic programming* (GP). We use the six-multiplexer problem as a means to B1.5.1 treat the validity of MDL-based fitness functions. Decision trees were proposed by Quinlan for concept

formation in machine learning (Quinlan 1983, 1986). Generating efficient decision trees from preclassified (supervised) training examples has generated a large literature. Decision trees can be used to represent Boolean concepts. Figure C4.4.1 shows a desirable decision tree which parsimoniously solves the six-multiplexer problem. In the six-multiplexer problem, a_0, and a_1 are the multiplexer addresses and d_0, d_1, d_2, and d_3 are the data. The target concept is

$$\text{output} = \overline{a_0}\,\overline{a_1}d_0 + a_0\overline{a_1}d_1 + \overline{a_0}a_1d_2 + a_0a_1d_3. \tag{C4.4.4}$$

A decision tree is a representation of classification rules: for example, the subtree on the left in figure C4.4.1 shows that the output becomes false (i.e. zero) when the variable $a_0 = 0$, $a_1 = 0$, and $d_0 = 0$.

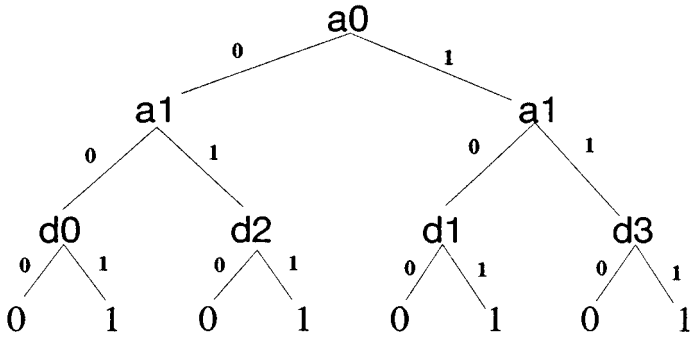


Figure C4.4.1. A more desirable and parsimonious decision tree.

Koza discussed the evolution of decision trees within a GP framework and conducted a small experiment called a *Saturday morning problem* (Koza 1990). However, Koza-style simple GP fails to evolve effective decision trees because an ordinary fitness function fails to consider parsimony. To overcome this shortcoming, we introduce fitness functions based on an MDL principle. This MDL-based fitness definition involves a tradeoff between the details of the tree, and the errors. In general, the MDL fitness definition for a GP tree (whose numerical value is represented by MDL) is defined as

$$\text{MDL} = (\text{exception\_coding\_length}) + (\text{tree\_coding\_length}) \tag{C4.4.5}$$

where exception_coding_length is the description length of residual error (i.e. the first term in equation (C4.4.1)) and tree_coding_length is the description length of the model.

The MDL value of a decision tree is calculated using the following method (Quinlan 1989). Consider the decision tree in figure C4.4.2 for the six-multiplexer problem (X, Y, and Z notations are explained later).

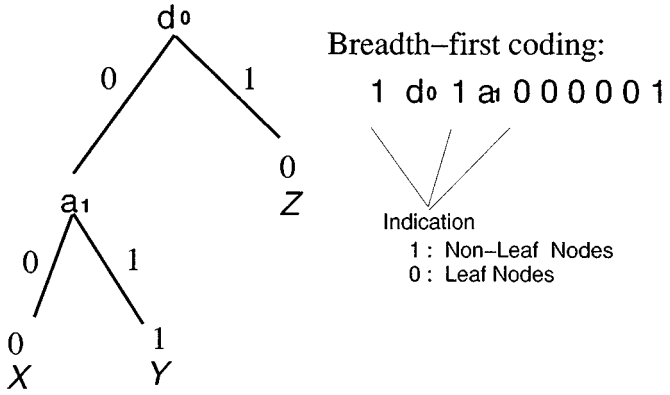


Figure C4.4.2. A decision tree for the six-multiplexer problem.

Using a breadth-first traversal, this tree is represented by the following string:

$$1\ d_0\ 1\ a_1\ 0\ 0\ 0\ 0\ 0\ 1. \tag{C4.4.6}$$

Since in our decision trees left (right) branches always represent zero (one) values for attribute-based tests, we can omit these attribute values. To encode this string in binary format,

$$2 + 3 + 2 \times \log_2 6 + 3 \times \log_2 2 = 13.17 \text{ bits} \tag{C4.4.7}$$

are required since the codes for each nonleaf (d_0, a_1) and for each leaf (0 or 1) require $\log_2 6$ and $\log_2 2$ bits respectively, and $2 + 3$ bits are used for their indications. In order to code exceptions (i.e. errors or incorrect classifications), their positions should be indicated. For this purpose, we divide the set of objects into classified subsets. For the tree of figure C4.4.2, we have three subsets (which we call X, Y, and Z from left to right) as shown in table C4.4.1. For instance, X is a subset whose members are classified into the leftmost leaf ($d_0 = 0 \ \wedge \ a_1 = 0$). The number of elements belonging to X is 16. 12 members of X are correctly classified (i.e. 0). Misclassified elements (i.e. four elements of X, eight elements of Y, and 20 elements of Z) can be coded with the following cost:

$$L(16, 4, 16) + L(16, 8, 8) + L(32, 20, 32) = 65.45 \text{ bits} \tag{C4.4.8}$$

where

$$L(n, k, b) = \log_2(b + 1) + \log_2 \left(\binom{n}{k} \right). \tag{C4.4.9}$$

$L(n, k, b)$ is the total cost for transmitting a bitstring of length n, in which k of the symbols are ones and b is an upper bound on k. Thus the total cost for the decision tree in figure C4.4.2 is 78.62 (=65.45+13.17) bits.

Table C4.4.1. Classified subsets for encoding exceptions.

Name	Attributes	No of elements	No of correct cl.	No of incorrect cl.
X	$d_0 = 0 \ \wedge \ a_1 = 0$	16	12	4
Y	$d_0 = 0 \ \wedge \ a_1 = 1$	16	8	8
Z	$d_0 = 1$	32	12	20

In general, the coding length for a decision tree with n_f attribute nodes and n_t terminal nodes is given as follows:

$$\text{tree\_coding\_length} = (n_\mathrm{f} + n_\mathrm{t}) + n_\mathrm{t} \log_2 T_\mathrm{s} + n_\mathrm{f} \log_2 F_\mathrm{s} \tag{C4.4.10}$$

$$\text{exception\_coding\_length} = \sum_{x \in \text{Terminals}} L(n_x, w_x, n_x) \tag{C4.4.11}$$

where T_s and F_s are the total numbers of terminals and functions respectively. In equation (C4.4.1), summation is taken over all terminal nodes. n_x is the number of elements belonging to the subset represented by x. w_x is the number of misclassified elements of n_x members.

With these preparations, we present results of the experiments to evolve decision trees for the six-multiplexer problem using GP. Table C4.4.2 shows the parameters used. A 1 (0) value in the terminal set represents a positive (negative) example, that is, a true (false) value. Symbols in the nonterminal set are attribute-based test functions. For the sake of explanation, we use S-expressions to represent decision trees from now on. The S-expression $(X \ Y \ Z)$ means that if X is 0 (false) then test the second argument Y and if X is 1 (true) then test the third argument Z. For instance, $(a_0 \ (d_1 \ (0) \ (1)) \ (1))$ is a decision tree which expresses that if a_0 is 0 (false) then if d_1 is 0 (false) then 0 (false) else 1 (true), and that if a_0 is 1 (true) then 1 (true).

Table C4.4.2. GP parameters.

Population size	100
Probability of graph crossover	0.6
Probability of graph mutation	0.0333
Terminal set	$\{0, 1\}$
Non-terminal set	$\{a_0, a_1, d_0, d_1, d_2, d_3\}$

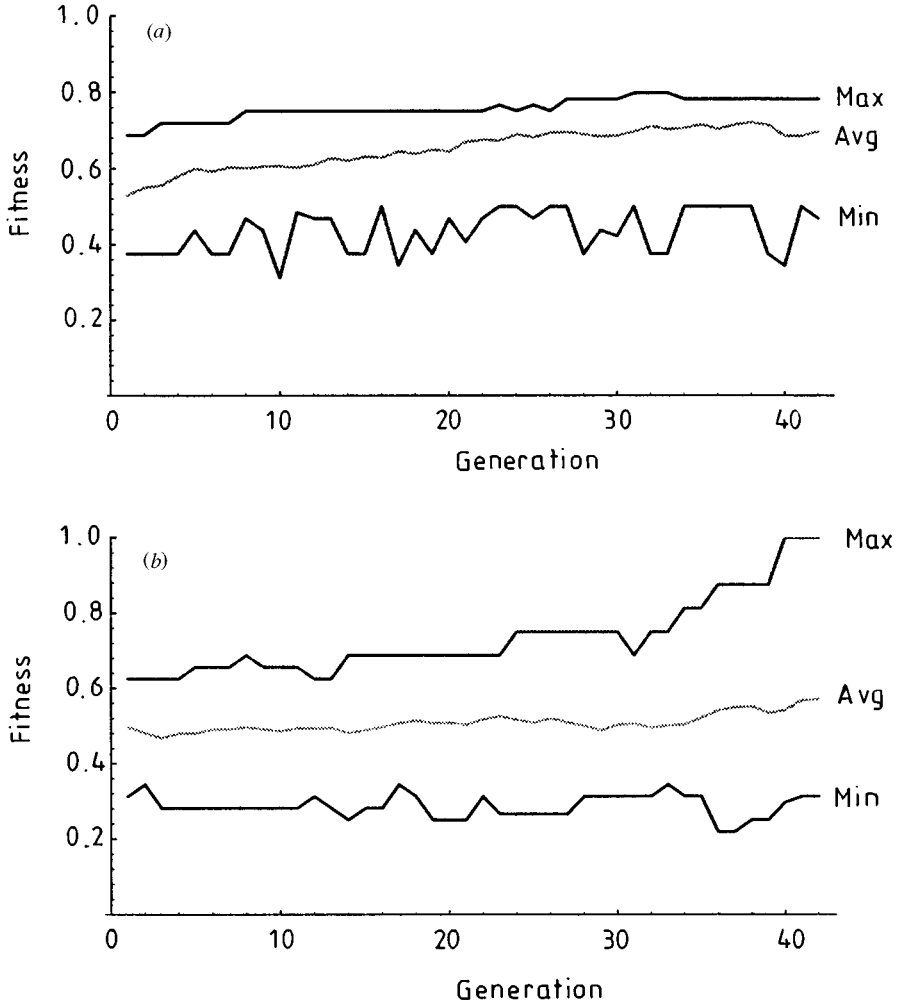


Figure C4.4.3. Experimental result: fitness versus generations.

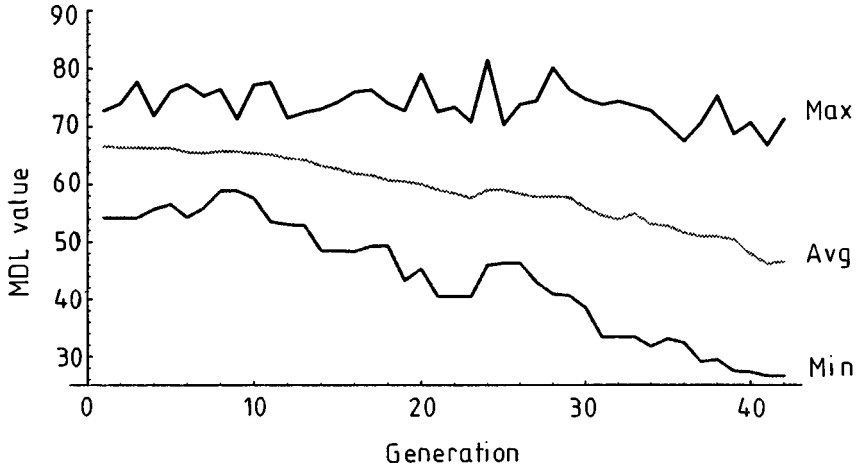


Figure C4.4.4. Experimental result: MDL versus generations.

Figure C4.4.3 shows results of experiments in terms of correct classification rate versus generations, using a traditional (non-MDL) fitness function (*a*), and using an MDL-based fitness function (*b*), where the traditional (non-MDL) fitness is defined as the rate of correct classification. The desired decision tree was acquired at the 40th generation when using an MDL-based fitness function. However, the largest fitness value (i.e. the rate of correct classification) at the 40th generation when using a non-MDL fitness function was only 78.12%. Figure C4.4.4 shows the evolution of the MDL values in the same experiment as figure C4.4.3(*b*). Figure C4.4.3(*a*) indicates clearly that the fitness test used in the non-MDL case is not appropriate for the problem. This certainly explains the lack of success in the non-MDL example. The acquired structure at the 40th generation using an MDL-based fitness function was as follows:

```
(A0
   (A1
      (D0
         (A1 (D0 (0) (0)) (D2 (0) (A0 (1) (1))))
         (1))
      (D2 (0) (1)))
   (A1 (D1 (0) (1)) (D3 (0) (1))))
```

whereas the typical genotype at the same generation (40th) using a non-MDL fitness function was as follows:

```
(D1
   (D2
      (D3 (0)
         (A0 (0)
            (D3 (A0 (0) (0))
               (D0 (D0 (1) (D1 (A0 (0) (1)) (D0 (D0 (0) (0)) (A1 (1) (1)))))
                  (D1 (0) (0))))))
      (A0
         (A0
            (D1 (1)
               (D1 (A0 (D1 (1) (1)) (A0 (D0 (D2 (1) (D1 (D1 (0) (0)) (1))) (0)) (1)))
                  (0)))
            (A0
               (A0
                  (D2
                     (A1 (1)
                        (D0
                           (D3 (0)
                              (A1 (D0 (A0 (1) (1)) (D3 (D0 (1) (0)) (A0 (0) (1))))
                                 (A1
                                    (D2
                                       (D2 (D0 (D2 (A1 (D3 (0) (D3 (0) (0))) (A1 (0) (1)))
                                                   (D3 (D3 (0) (0)) (1)))
                                               (D2 (1) (D0 (1) (0))))
                                           (0))
                                       (D0 (D3 (D2 (A0 (D3 (0) (0)) (1)) (1)) (1)) (1)))
                                    (1))))
                           (D0 (D3 (A0 (1) (A0 (0) (A1 (0) (A0 (1) (1))))) (D3 (1) (0))) (1))))
                     (1))
                  (A0 (A0 (0) (0)) (A1 (0) (D0 (1) (1)))))
               (A1
                  (A0 (1)
                     (D3 (D1 (D2 (D1 (0) (A0 (1) (D3 (D1 (0) (1)) (1)))) (0)) (1)) (1)))
                  (0))))
         (0)))
   (D2 (A1 (A1 (1) (0)) (A1 (1) (0))) (1))).
```

As can be seen, the non-MDL fitness function did not control the growth of the decision trees, whereas using an MDL-based fitness function led to a successful learning of a satisfactory data structure. Thus we can conclude that an MDL-based fitness function works well for the six-multiplexer problem.

C4.4.4 Recent studies on complexity-based fitness

The complexity-based fitness evaluation can be introduced in order to control *genetic algorithm* (GA) search strategies. For instance, when applying GAs to the classification of genetic sequences, Konagaya and Konoto (1993) employed the MDL principle for GA fitness in order to avoid overlearning caused by the statistical fluctuations. They presented a GA-based methodology for learning stochastic motifs from given genetic sequences. A stochastic motif is a probabilistic mapping from a genetic sequence (which has been drawn from a finite alphabet) to a number of categories (such as cytochrome, globin, and trypsin). They employed Rissanen's MDL principle in selecting an optimal hypothesis (Yamanishi and Konagaya 1991). B1.2

When applying the MDL principle to GP, redundant structures should be pruned as much as possible, but at the same time premature convergence (i.e. premature loss of genotypic diversity) should be avoided (Zhang and Mühlenbein 1995). Zhang and Mühlenbein proposed a dynamic control to fix the error factor at each generation and change the complexity factor adaptively with respect to the error. Let $E_i(g)$ and $C_i(g)$ denote the error and complexity of the ith individual at generation g. Assuming that $0 \leq E_i(g) \leq 1$ and $C_i(g) \geq 0$, they defined the fitness of an individual i at generation g as

$$F_i(g) = E_i(g) + \alpha(g)C_i(g) \tag{C4.4.12}$$

where $\alpha(g)$ is called the adaptive Occam factor and is expressed as

$$\alpha(g) = \begin{cases} \dfrac{1}{N^2}\dfrac{E_{\text{best}}(g-1)}{\check{C}_{\text{best}}(g)} & \text{if } E_{\text{best}}(g-1) > \epsilon \\ \dfrac{1}{N^2}\dfrac{1}{E_{\text{best}}(g-1)\check{C}_{\text{best}}(g)} & \text{otherwise} \end{cases} \tag{C4.4.13}$$

where N is the size of the training set, E_{best} is the error value of the program which has the smallest (best) fitness value at generation $g-1$, and $\check{C}_{\text{best}}(g)$ is the size of the best program at generation g estimated at generation $g-1$, which is used for the normalization of the complexity factor. The user-defined constant ϵ specifies the maximum training error allowed for the final solution. Zhang and Mühlenbein have shown experimental results in the GP of sigma–pi neural networks. Their results were satisfactory.

In the articles by Iba and coworkers (1993, 1994), MDL-based fitness functions were applied successfully to *system identification problems* by using the implemented system STROGANOFF. The results showed that MDL-based fitness evaluation works well for tree structures in STROGANOFF, which controls GP-based tree search (see G1.7 for more details). F1.4

C4.4.5 Conclusion

To conclude this section, we have shown that complexity-based fitness evaluations work by introducing a penalty term for the model complexity. We described several model selection criteria proposed so far. However the advantages and disadvantages of these approaches are not clear and are still being debated (see the article by Rissanen (1987) for details). Their different theoretical backgrounds and philosophies make it difficult to conduct comparative studies. The applicability and robustness of these methods remain to be seen as an interesting future topic.

References

Akaike H 1977 On the entropy maximization principle *Applications of Statistics* ed P R Krishnaiah (Amsterdam: North-Holland)

Barron A R 1991 Minimum complexity density estimation *IEEE Trans. Information Theory* **IT-37** 1034–54

Iba H, Kurita T, deGaris H and Sato T 1993 System identification using structured genetic algorithms *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 279–86

Iba H, deGaris H and Sato T 1994 Genetic programming using a minimum description length principle *Advances in Genetic Programming* ed K E Kinnear Jr (Cambridge, MA: MIT Press) pp 265–84

Konagaya A and Kondo H 1993 Stochastic motif extraction using a genetic algorithm with the MDL principle *Hawaii Int. Conf. on Computer Systems (1993)*

Koza J 1990 *Genetic Programming a Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems* Stanford University, Department of Computer Science, Report STAN-CS-90-1314

Quinlan J R 1983 Learning efficient classification procedures and their application to chess end games *Machine Learning* ed R S Michalski, J G Carbonell and T M Mitchell (Berlin: Springer) pp 463–82

——1986 Induction of decision trees *Machine Learning* **1** 81–106

——1989 Inferring decision trees using the minimum description length principles *Information Comput.* **80** 227–48

Rissanen J 1987 Stochastic complexity *J. R. Stat. Soc.* B **49** pp 223–39, 252–65

——1989 *Stochastic Complexity in Statistical Inquiry* (Singapore: World Scientific)

Seshu R 1994 Binary decision trees and an 'average-case' model for concept learning: implications for feature construction and the study of bias *Computational Learning Theory and Natural Learning Systems, Volume I: Constraints and Prospects* ed J Hanson *et al* (Cambridge, MA: MIT Press) pp 213–48

Tenorio M F and Lee W 1990 Self-organizing network for optimum supervised learning *IEEE Trans. Neural Networks* **NN-1** 100–9

Wallace C S and Freeman P R 1987 Estimation and inference by compact coding *J. R. Stat. Soc.* B **49** 240–65

Weigend A S and Rumelhart D E 1994 Weight elimination and effective network size *Computational Learning Theory and Natural Learning Systems, Volume I: Constraints and Prospects* ed J Hanson *et al* (Cambridge, MA: MIT Press) pp 457–76

Yamanishi K and Konagaya A 1991 Learning stochastic motifs from genetic sequences *Machine Learning (Proc. 8th Int. Workshop)* ed L A Birnbaum and G C Collins (San Mateo, CA: Morgan Kaufmann) pp 467–71

Yamanishi K A 1992 Learning criterion for stochastic rules *Machine Learning* (**9** 165–204

Zhang B-T and Mühlenbein H 1995 Balancing accuracy and parsimony in genetic programming *Evolutionary Comput.* **3** 17–38

C4.5 Multiobjective optimization

C M Fonseca and P J Fleming

Abstract

This section reviews current approaches to multiobjective optimization with evolutionary algorithms. After introducing the subject, multiobjective fitness evaluation is formulated as a two-step process: a decision-making problem which reflects the preferences of an expert in the relevant problem domain followed by conventional fitness assignment and selection. The presentation and discussion of the various methods proposed in the literature follows, with a preference for concept over chronology.

C4.5.1 Introduction

Real-world problems often involve multiple measures of performance, or objectives, which should be optimized simultaneously. In practice, however, this is not always possible, as some of the objectives may be conflicting. Objectives are often also *noncommensurate*; that is, they measure fundamentally different aspects of the quality of a candidate solution. Thus, the quality of an individual is better described, not by a scalar, but by a vector quantity.

Performance, reliability, and cost are typical examples of conflicting, noncommensurate objectives. Improvement in any combination of these objectives will unequivocally improve the overall solution, but only as long as no degradation occurs in the remaining objectives. If this is not possible, then the current solution is said to be optimal in the *Pareto* sense, Pareto optimal, or nondominated. Otherwise, the new solution is said to dominate the old one. The set of all Pareto-optimal solutions is known as the Pareto-optimal set. F1.9.3.2

In most practical cases, a single compromise solution is sought. Thus, multiobjective optimization is generally more than purely searching for Pareto-optimal solutions. To be able to produce acceptable solutions, multiobjective optimization methods also need to take into account human preferences. In fact, although a Pareto-optimal solution should always be a better compromise solution than any solution it dominates, not all Pareto-optimal solutions may constitute acceptable compromise solutions.

C4.5.2 Fitness evaluation

Multiobjective optimization with evolutionary algorithms, as with other optimizers, must ultimately be based on a scalarization of the objectives. Fitness, as a measure of the expected number of offspring of an individual, must remain a scalar. This scalarization should be a coordinatewise monotonic transformation, *but not necessarily a function*, of the objectives, so that individuals are always guaranteed to be at least as fit as those they dominate in the Pareto sense. Such a transformation, being clearly nonunique, provides the necessary scope for incorporating preference information in the rating of the solutions. Once a scalar measure of quality (or cost) has been derived, the evolutionary algorithm may proceed with fitness assignment and selection as usual.

The cost assignment problem with multiple objectives is, essentially, a decision-making problem involving a finite number of objects, i.e. the individuals in the population, given what knowledge of the problem is available at the time of the decision. In this context, if a good decision strategy has been developed for a particular multiobjective problem, it should be possible to base the corresponding evaluation of fitness on that same strategy.

C4.5.3 Current evolutionary approaches to multiobjective optimization

Current approaches to multiobjective optimization with evolutionary algorithms may be divided into three groups (Fonseca and Fleming 1995). (See also Section F1.9 of this handbook.) F1.9

- *Plain aggregating approaches.* Objectives are numerically combined into a single objective *function* to be optimized.
- *Population-based non-Pareto approaches.* Different objectives affect the selection or deselection of different parts of the population in turn. Approaches based on separate rankings of the population according to each objective also fit in this category.
- *Pareto-based approaches.* The population is ranked making direct use of the definition of Pareto dominance.

Given the diversity of the approaches proposed in the literature to date, this classification is necessarily a rough one. However, it does reflect three of the main ideas behind the current handling of multiple objectives in evolutionary optimization, as the following review documents. Minimization of the objectives is assumed throughout except where noted otherwise.

C4.5.3.1 The weighted-sum approach

Working mechanism. The n objectives $f_1, \ldots, f_n$ are weighted by user-defined, positive coefficients $w_1, \ldots, w_n$ and added together to obtain a scalar measure of cost for each individual. This measure can then be used as the basis for selection, e.g. *proportional*, *tournament*, or based on *ranking*. C2.2, C2.3

The weighted-sum approach is widely known, intuitive and simple to implement, and can be used with virtually all optimizers. Consequently, it is also the most popular. C2.4

Formal description.

$$\Phi : \mathbb{R}^n \longrightarrow \mathbb{R}$$
$$\boldsymbol{f}(\boldsymbol{a}_i) \longmapsto \sum_{k=1}^{n} w_k \, f_k(\boldsymbol{a}_i)$$

where Φ denotes the cost assignment strategy.

Parameter settings. The setting of the weighting coefficients w_k is generally dependent on the problem instance, and not just on the problem class. Thus, the initial combination of weights usually needs to be fine tuned in order to lead to satisfactory compromise solutions. This usually implies rerunning the optimizer, although it may also be possible to modify the weights as the evolutionary algorithm runs (see Section C2.9). Hajela and Lin (1992) encoded the weights at the chromosome level, and promoted their diversity through *sharing*. C2.9

Theory. For any set of positive weights, the (global) optimum of Φ is always a nondominated solution of the original multiobjective problem (Goicoechea *et al* 1982). However, the opposite is not true. For example, nondominated solutions located in concave regions of the tradeoff surface cannot be obtained by this method, because their corresponding value of Φ is suboptimal (see, for example, the article by Fleming and Pashkevich (1985)). This is also illustrated in figure C4.5.1.

C4.5.3.2 The minimax approach

Working mechanism. This approach consists of minimizing the maximum of the n objectives $f_1, \ldots, f_n$. In practice, it is often implemented as the minimization of the maximum (weighted) difference between the objectives and goals $g_1, \ldots, g_n$ specified by the user for each objective. Wilson and McLeod (1993) see this approach as a form of goal attainment (Gembicki 1974).

Formal description.

$$\Phi : \mathbb{R}^n \longrightarrow \mathbb{R}$$
$$\boldsymbol{f}(\boldsymbol{a}_i) \longmapsto \max_{k=1,\ldots,n} \frac{f_k(\boldsymbol{a}_i) - g_k}{w_k}.$$

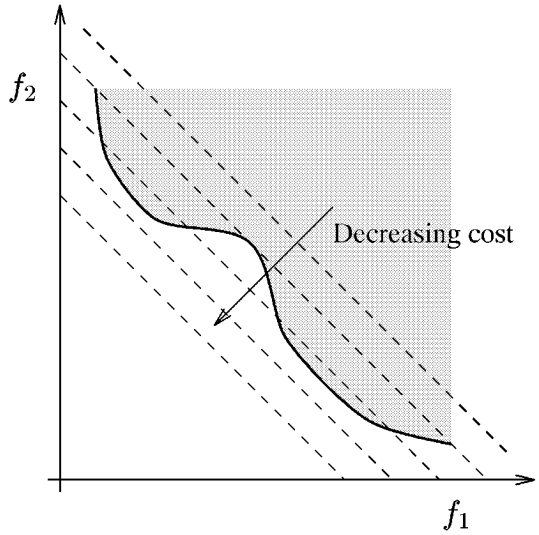


Figure C4.5.1. Lines of equal cost Φ induced by the weighted-sum approach (Fonseca 1995).

Parameter settings. The goal values g_k indicate levels of performance in each objective dimension k to be approximated and, if possible, improved upon by the final solution. In practice, the goals are often set to the desired levels of performance or, alternatively, to Utopian values known *a priori* to be unattainable.

The weights w_k indicate the desired direction of search in objective space, and are often set to the absolute value of the goals. The smaller a weight, the *harder* the corresponding objective becomes with respect to the remaining objectives. Hard objectives are essentially constraints, in that the corresponding goals must be attained, but only by a minimal amount.

Theory. The minimax approach usually results in a cost function with regions of nonsmoothness, typically including the optimum, even if the objective functions themselves are smooth. For this reason, alternative formulations such as the goal attainment method (Gembicki 1974) are usually preferred to this approach when gradient-based optimizers are used. However, since evolutionary algorithms do not usually use gradient information, this should raise no concern.

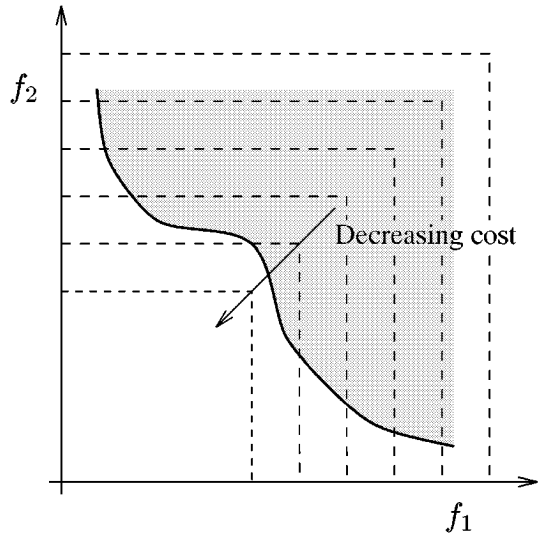


Figure C4.5.2. Lines of equal cost Φ induced by the minimax approach (Fonseca 1995).

Although this approach is able to produce solutions in concave regions of the tradeoff surface (see figure C4.5.2) the minimization of Φ is not guaranteed to produce strictly nondominated solutions. In fact, it is easy to show that one solution may dominate another, and yet have the same cost Φ.

C4.5.3.3 The target vector approach

Working mechanism. This approach consists of minimizing the distance of the objective vector $\boldsymbol{f} = (f_1, \ldots, f_n)$ from a predefined goal, or target, vector $\boldsymbol{g} = (g_1, \ldots, g_n)$, according to a suitable distance measure (Wienke *et al* 1992).

Formal description.

$$\begin{aligned} \Phi : \mathbb{R}^n &\longrightarrow \mathbb{R} \\ \boldsymbol{f}(\boldsymbol{a}_i) &\longmapsto \left\| [\boldsymbol{f}(\boldsymbol{a}_i) - \boldsymbol{g}] \mathbf{W}^{-1} \right\|_\alpha . \end{aligned}$$

Parameter settings. The goal values $g_1, \ldots, g_n$ indicate the desired levels of performance in each objective dimension, which are to be approximated by the final solution as closely as possible, typically in a weighted Euclidean sense ($\alpha = 2$). The weighting matrix $\mathbf{W}$ is often chosen to be diagonal, but in specific applications more elaborate weighting schemes may be appropriate (Wienke *et al* 1992).

C4.5.3.4 The lexicographic approach

Working mechanism. Objectives are assigned distinct priorities according to how important they are, prior to optimization. The objective with the highest priority is used first when comparing individuals, either to decide a *tournament* (Fourman 1985) or to *rank* the population in a single-objective fashion. Any ties are resolved by comparing the relevant individuals again with respect to the second-highest-priority objective, and so forth, until the lowest-priority objective is reached. C2.3, C2.4

Formal description.

$$\begin{aligned} \Phi : \mathbb{R}^n &\longrightarrow \{0, 1, \ldots, \mu - 1\} \\ \boldsymbol{f}(\boldsymbol{a}_i) &\longmapsto \sum_{j=1}^{\mu} \mathbb{1}\left(\boldsymbol{f}(\boldsymbol{a}_j) \prec \boldsymbol{f}(\boldsymbol{a}_i)\right) \end{aligned}$$

where $\boldsymbol{f}(\boldsymbol{a}_j) \prec \boldsymbol{f}(\boldsymbol{a}_i)$ if and only if

$$\exists\, p \in \{1, \ldots, n\} : \ \forall k \in \{p, \ldots, n\} \qquad f_k(\boldsymbol{a}_j) \leq f_k(\boldsymbol{a}_i) \ \wedge \ f_p(\boldsymbol{a}_j) < f_p(\boldsymbol{a}_i)$$

and where $\mathbb{1}$(condition) evaluates to unity if the condition is verified, and to zero otherwise.

The objectives $f_1, \ldots, f_n$ are assumed to be sorted in order of increasing priority.

Parameter settings. All objectives must be assigned distinct priorities. This requirement will be acceptable in those practical situations where decisions regarding the quality of a solution are made sequentially (Ben-Tal 1980), and where the relative importance of the various objectives is well understood.

In the case of heavily competing objectives, the final solution may vary wildly depending on how priorities are assigned. For example, if all objectives admit single, but different, optima, the lexicographic optimum will be no more than the optimum of the highest-priority objective.

Theory. Lexicographically optimal solutions are also, by definition, Pareto-optimal solutions.

C4.5.3.5 The VEGA approach

Working mechanism. The *vector-evaluated genetic algorithm* (VEGA, Schaffer 1985) was probably the first evolutionary approach explicitly aimed at promoting the generation of multiple nondominated solutions. Subpopulations of offspring are selected according to each objective in turn, using *fitness-proportionate* selection. Offspring are then mixed in order to undergo recombination and mutation, regardless of which objective dictated their selection. F1.9, C2.2

Formal description. The merging of the offspring subpopulations in VEGA is equivalent to averaging the fitness components δ_k corresponding to each objective f_k (here maximization is assumed).

$$\Delta : \mathbb{R}^n \longrightarrow \mathbb{R}$$
$$\boldsymbol{f}(\boldsymbol{a}_i) \longmapsto \sum_{k=1}^{n} \lambda_k \, \delta_k \left[f_k(\boldsymbol{a}_i) \right]$$

where, since proportional selection is used,

$$\delta_k \left[f_k(\boldsymbol{a}_i) \right] = \frac{f_k(\boldsymbol{a}_i)}{\sum_{j=1}^{\mu} f_k(\boldsymbol{a}_j)} .$$

Note that, whereas Φ has been used earlier to denote a cost assignment strategy on which selection can be based, Δ is used here to denote a multiobjective *fitness* assignment strategy, in analogy with the use of δ to denote a single-objective scaling function.

Parameter settings. The size of each subpopulation, λ_k, controls the involvement of the associated objective in the selection process. In Schaffer's original work (Schaffer 1985), all subpopulations were the same size.

Theory. On concave tradeoff surfaces, the population may split into *species* particularly strong in each objective (*speciation*, Schaffer 1985). This undesirable effect has been noted to arise from VEGA ultimately performing a weighted sum of the objectives (Richardson *et al* 1989), even though the weights associated with this linear combination depend on the distribution of the population at each generation.

However, by effectively weighting each objective proportionally to the inverse of the total population fitness in that objective dimension, VEGA adaptively attempts to balance improvement in the various objective dimensions. If improvement is observed only in some objectives, selection will then favor improvement in the remaining objectives. As a result, VEGA can, at least in some cases, maintain different species much longer than a GA optimizing a pure weighted-sum would do, due to *genetic drift* A2.2.5 (Fonseca and Fleming 1995).

C4.5.3.6 The median-rank approach

Working mechanism. The population is first ranked according to each single objective, separately. Then, the median of the ranks assigned to each individual is computed and used for fitness assignment (Breeden 1995).

Variations of this approach include implementations where the average is used to replace the median, and where fitness values are computed based on each ranking first, and only subsequently averaged. Implementations where objectives are used in turn to decide tournaments (Fourman 1985), or to dictate the *deletion* of fractions of the population (Kursawe 1991), can also be seen as fitting in this category. C3.4.3

Formal description.

$$\Phi : \mathbb{R}^n \longrightarrow [0, \mu - 1]$$
$$\boldsymbol{f}(\boldsymbol{a}_i) \longmapsto \text{median}\{\text{rank}_1[f_1(\boldsymbol{a}_i)], \ldots, \text{rank}_n[f_n(\boldsymbol{a}_i)]\}.$$

Parameter settings. The plain median-rank approach implicitly assumes that all objectives are equally important. The median analogue of a weighted sum can nevertheless be achieved by entering the rank value associated with each objective a number of times proportional to its importance, and computing the median of the resulting sample (Breeden 1995).

Computing the average instead of the median may offer a simpler way of controlling the relative importance of each objective, but the resulting algorithm will also be more sensitive to any uncertainty in the evaluation of the objective functions.

Theory. Ranking the population according to each objective separately avoids the normalization difficulties associated with all aggregating function approaches. As a consequence, and despite the similarity to the weighted-sum approach, algorithms based on these methods are not affected by whether tradeoff surfaces are convex or concave.

C4.5.3.7 Pareto ranking

Working mechanism. The concept of Pareto dominance is used to rank the population in such a way that all nondominated individuals in the population are assigned the same cost. Two approaches to Pareto ranking have been proposed in the literature.

(i) In the approach originally proposed by Goldberg (1989), all nondominated individuals in the population are assigned a cost of one, and removed from contention. Then, a new set of nondominated individuals is identified and assigned a cost of two. The process continues until the whole population has been ranked.

(ii) An alternative approach has been proposed by Fonseca and Fleming (1993), where individuals are simply assigned a cost value according to how many individuals in the population they are dominated by.

Once computed, these cost values are used to perform selection, e.g. using rank-based fitness assignment (Fonseca and Fleming 1993, Srinivas and Deb 1994), or tournament selection (Cieniawski 1993, Ritzel *et al* 1994). Rather than ranking the population first, Horn *et al* (1994) based their tournament selection directly on whether or not one of the competitors dominated the other one.

Goldberg (1989) has also noted the need for *niching* and *speciation* techniques in order to stabilize the population along the Pareto front. Most of the implementations of Pareto-based fitness assignment cited above also include techniques such as *fitness sharing* and *mating restriction*. C6.1, C6.2 C6.1.2, C6.2.4

Formal description.

(i) The first ranking strategy will be defined through recursion:

$$\Phi : \mathbb{R}^n \longrightarrow \{1, 2, \ldots, \mu\}$$

$$\boldsymbol{f}(\boldsymbol{a}_i) \longmapsto \begin{cases} 1 & \Leftarrow \quad \neg[\boldsymbol{f}(\boldsymbol{a}_j) \text{ p}\prec \boldsymbol{f}(\boldsymbol{a}_i)] \quad \forall j \in \{1, \ldots, \mu\} \\ \\ \phi & \Leftarrow \quad \neg[\boldsymbol{f}(\boldsymbol{a}_j) \text{ p}\prec \boldsymbol{f}(\boldsymbol{a}_i)] \quad \forall j \in \{1, \ldots, \mu\} \setminus \{l : \Phi[\boldsymbol{f}(\boldsymbol{a}_l)] < \phi\} \end{cases}$$

where $\boldsymbol{f}(\boldsymbol{a}_j)$ p$\prec$ $\boldsymbol{f}(\boldsymbol{a}_i)$ if and only if

$$\forall k \in \{1, \ldots, n\} \qquad f_k(\boldsymbol{a}_j) \leq f_k(\boldsymbol{a}_i) \ \wedge \ \exists k \in \{1, \ldots, n\} : \ f_k(\boldsymbol{a}_j) < f_k(\boldsymbol{a}_i)$$

and where the symbol $\neg$ denotes logical negation.

(ii) The second ranking strategy follows is simpler to define:

$$\Phi : \mathbb{R}^n \longrightarrow \{0, 1, \ldots, \mu - 1\}$$

$$\boldsymbol{f}(\boldsymbol{a}_i) \longmapsto \sum_{j=1}^{\mu} \mathbb{1}\left(\boldsymbol{f}(\boldsymbol{a}_j) \text{ p}\prec \boldsymbol{f}(\boldsymbol{a}_i)\right)$$

where $\mathbb{1}$(condition) evaluates to unity if the condition is verified and to zero otherwise.

Parameter settings. There are no parameters to set. This is especially attractive if the relative importance of the different objectives is not known *a priori*, or cannot be formally expressed easily. Unfortunately, this also means that, even if preference information is indeed available, it cannot be used.

Other than implementation issues, there is currently no reason to prefer one ranking strategy to the other. The second approach seems to be easier to interpret in the (theoretical) infinite-population case (Fonseca and Fleming 1995), and thus may be easier to analyze.

Theory. Both ranking procedures are such that $\boldsymbol{f}(\boldsymbol{a}_j)$ p$<$ $\boldsymbol{f}(\boldsymbol{a}_i)$ implies $\Phi[\boldsymbol{f}(\boldsymbol{a}_j)] < \Phi[\boldsymbol{f}(\boldsymbol{a}_i)]$, and that all nondominated individuals are assigned the same cost. As a consequence, it is possible for the population to stagnate if it enters a state where most individuals are nondominated. This is especially likely to happen as the number of competing objectives increases.

The ranks assigned by method (ii) to a large uniformly distributed population, once normalized by the population size, may be seen as estimates of what fraction of the search space outperforms each particular point considered. For problems involving two decision variables only, this interpretation of ranking allows the visualization of the cost landscapes induced by Pareto ranking, as well as by ranking based on other concepts, such as lexicographic optimality (discussed earlier), and *preferability* given a goal vector, which will be discussed next.

C4.5.3.8 Pareto-like ranking with goal and priority information

Working mechanism. In this approach, a concept which combines Pareto dominance with goal and priority information is used instead of pure Pareto dominance to rank the population by the second method described above, making it possible to bias the search away from regions of the tradeoff surface known *a priori* to be unacceptable.

As in the lexicographic approach, higher-priority objectives come into play before those with a lower priority, but different objectives may now be assigned equal priorities. In addition to this, the comparison of solutions is affected by whether or not they attain the goals set for the various objectives.

Since the setting of goals and priorities ultimately depends on the personal preference of the operator, this concept has been called *preferability* (Fonseca 1995).

Formal description.

$$\begin{aligned} \Phi : \mathbb{R}^n &\longrightarrow \{0, 1, \ldots, \mu - 1\} \\ \boldsymbol{f}(\boldsymbol{a}_i) &\longmapsto \sum_{j=1}^{\mu} \mathbb{1}\left(\boldsymbol{f}(\boldsymbol{a}_j) \underset{\boldsymbol{g}}{\prec} \boldsymbol{f}(\boldsymbol{a}_i)\right) \end{aligned}$$

where the symbol $\underset{\boldsymbol{g}}{\prec}$ indicates preferability given the *preference vector* $\boldsymbol{g}$.

To define preferability, consider the n-dimensional preference vector $\boldsymbol{g}$, where n is the number of objectives, as a concatenation of p vectors $\boldsymbol{g}_m$, $m = 1, \ldots, p$. Each subvector $\boldsymbol{g}_m$ contains those n_m components of $\boldsymbol{g}$ which have been assigned priority m, such that

$$\sum_{m=1}^{p} n_m = n.$$

Also, let the smile $\overset{j}{\smile}$ index the components of $\boldsymbol{f}$ and $\boldsymbol{g}$ where $\boldsymbol{f}(a_j) \preceq \boldsymbol{g}$, and let the frown $\overset{j}{\frown}$ index the remaining components of these vectors, for each given individual a_j, and similarly for $\boldsymbol{f}_m$ and $\boldsymbol{g}_m$, $m = 1, \ldots, p$. Then, $\boldsymbol{f}(\boldsymbol{a}_j) \underset{\boldsymbol{g}}{\prec} \boldsymbol{f}(\boldsymbol{a}_i)$ if and only if

$$\begin{aligned} p = 1 \Rightarrow \left(\boldsymbol{f}_p(\boldsymbol{a}_j)^{\overset{j}{\frown}} \text{ p}< \boldsymbol{f}_p(\boldsymbol{a}_i)^{\overset{j}{\frown}}\right) \vee \Bigg\{ &\left(\boldsymbol{f}_p(\boldsymbol{a}_j)^{\overset{j}{\frown}} = \boldsymbol{f}_p(\boldsymbol{a}_i)^{\overset{j}{\frown}}\right) \\ &\wedge \left[\left(\boldsymbol{f}_p(\boldsymbol{a}_i)^{\overset{j}{\smile}} \not\preceq \boldsymbol{g}_p^{\overset{j}{\smile}}\right) \vee \left(\boldsymbol{f}_p(\boldsymbol{a}_j)^{\overset{j}{\smile}} \text{ p}< \boldsymbol{f}_p(\boldsymbol{a}_i)^{\overset{j}{\smile}}\right)\right]\Bigg\} \end{aligned}$$

and

$$\begin{aligned} p > 1 \Rightarrow \left(\boldsymbol{f}_p(\boldsymbol{a}_j)^{\overset{j}{\frown}} \text{ p}< \boldsymbol{f}_p(\boldsymbol{a}_i)^{\overset{j}{\frown}}\right) \vee \Bigg\{ &\left(\boldsymbol{f}_p(\boldsymbol{a}_j)^{\overset{j}{\frown}} = \boldsymbol{f}_p(\boldsymbol{a}_i)^{\overset{j}{\frown}}\right) \\ &\wedge \left[\left(\boldsymbol{f}_p(\boldsymbol{a}_i)^{\overset{j}{\smile}} \not\preceq \boldsymbol{g}_p^{\overset{j}{\smile}}\right) \vee \left(\boldsymbol{f}_{1,\ldots,p-1}(\boldsymbol{a}_j) \underset{\boldsymbol{g}_{1,\ldots,p-1}}{\prec} \boldsymbol{f}_{1,\ldots,p-1}(\boldsymbol{a}_i)\right)\right]\Bigg\} \end{aligned}$$

where $\boldsymbol{f}_{1,\ldots,p-1}$ denotes the concatenation of $\boldsymbol{f}_1, \ldots, \boldsymbol{f}_{p-1}$, and similarly for $\boldsymbol{g}_{1,\ldots,p-1}$.

Parameter settings. By setting goals and assigning priorities to the objectives, a number of other, simpler cost assignment strategies can be implemented (Fonseca 1995).

- *Pareto.* All objectives are assigned priority 1 and fully unattainable goals; that is,

$$\boldsymbol{g} = \boldsymbol{g}_1 = [(-\infty, \ldots, -\infty)].$$

- *Lexicographic.* Objectives are all assigned different priorities and fully unattainable goals; that is,

$$\boldsymbol{g} = (\boldsymbol{g}_1, \ldots, \boldsymbol{g}_n) = [(-\infty), \ldots, (-\infty)].$$

- *Constrained optimization.* Inequality constraints are handled as priority 2 objectives to be minimized until the corresponding goals are reached. Assuming Pareto optimization for the soft objectives, one has

$$\boldsymbol{g} = (\boldsymbol{g}_1, \boldsymbol{g}_2) = [(-\infty, \ldots, -\infty), (g_{2\,1}, \ldots, g_{2\,n_c})]$$

  where n_c is the number of constraints. If there are no soft objectives, the problem becomes a constraint satisfaction problem:

$$\boldsymbol{g} = (\boldsymbol{g}_2) = [(g_{2,1}, \ldots, g_{2,n_c})].$$

- *Goal programming.* Goals are set as for the minimax approach, and all objectives are assigned priority 1:

$$\boldsymbol{g} = (\boldsymbol{g}_1) = [(g_{1,1}, \ldots, g_{1,n})].$$

  These parameters would promote the sampling of the portion of the tradeoff surface which dominates the goals, if they are attainable, or, if they are unattainable, the region dominated by the goals.

Goals and priorities may also be set *interactively* during an optimization session (Fonseca and Fleming 1993, Fonseca 1995). C2.9

Theory. It can be shown that all nondominated solutions will also be preferred solutions for some setting of the preference vector $\boldsymbol{g}$. However, some preferred solutions may not be nondominated. The preferability relation can also be shown to be transitive. Detailed proofs can be found in the thesis of Fonseca (1995, pp 154–9).

C4.5.4 Concluding remarks

The number and diversity of the multiobjective approaches to evolutionary optimization proposed to date is a clear sign of the growing interest and recognition this area is receiving. In contrast, quantitatively characterizing their expected performance, even on specific examples, has remained difficult, mainly due to the lack of a unique solution to such problems and to the number of performance dimensions involved. Needless to say, this has considerably impaired the realization of extensive comparative studies.

In the light of recent results concerning the performance assessment and comparison of multiobjective optimizers such as, but not limited to, evolutionary algorithms (Fonseca and Fleming 1996), this situation may soon change. Until then, the choice of a multiobjective fitness evaluation approach should take into account how much preference information is available for a particular problem, and in what form, as well the ease of implementation and whether or not it is possible or desirable to interact with the algorithm as it runs.

References

Ben-Tal A 1980 Characterization of Pareto and lexicographic optimal solutions *Multiple Criteria Decision Making Theory and Application (Lecture Notes in Economics and Mathematical Systems 177)* pp 1–11

Breeden J L 1995 Optimizing stochastic and multiple fitness functions *Proc. 4th Annu. Conf. on Evolutionary Programming (San Diego, CA, 1995)* ed J R McDonnel, R G Reynolds and D B Fogel (Cambridge, MA: MIT Press)

Cieniawski S E 1993 *An Investigation of the Ability of Genetic Algorithms to Generate the Tradeoff Curve of a Multi-objective Groundwater Monitoring Problem* Master's Thesis, University of Illinois at Urbana-Champaign

Fleming P J and Pashkevich A 1985 Computer aided control system design using a multiobjective optimization approach *Proc. IEE Control'85 Conf. (Cambridge, 1985)* pp 174–9

Fonseca C M 1995 *Multiobjective Genetic Algorithms with Application to Control Engineering Systems* PhD Thesis, University of Sheffield

Fonseca C M and Fleming P J 1993 Genetic algorithms for multiobjective optimization: formulation, discussion and generalization *Genetic Algorithms: Proc. 5th Int. Conf. (Urbana-Champaign, IL, 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 416–23

——1995 An overview of evolutionary algorithms in multiobjective optimization *Evolutionary Comput.* **3** 1–16

——1996 On the performance assessment and comparison of stochastic multiobjective optimizers *Parallel Problem Solving from Nature—PPSN IV* ed H-M Voigt, W Ebeling, I Rechenberg and H-P Schwefel (Berlin: Springer) pp 584–93

Fourman M P 1985 Compaction of symbolic layout using genetic algorithms *Genetic Algorithms and Their Applications: Proc. 1st Int. Conf. on Genetic Algorithms* ed J J Grefenstette (Hillsdale, NJ: Erlbaum) pp 141–53

Gembicki F W 1974 *Vector Optimization for Control with Performance and Parameter Sensitivity Indices* PhD Thesis, Case Western Reserve University

Goicoechea A, Hansen D R and Duckstein L 1982 *Multiobjective Decision Analysis with Engineering and Business Applications* (New York: Wiley) p 46

Goldberg D E 1989 *Genetic Algorithms in Search, Optimization and Machine Learning* (Reading, MA: Addison-Wesley)

Hajela P and Lin C-Y 1992 Genetic search strategies in multicriterion optimal design *Struct. Optimization* **4** 99–107

Horn J, Nafpliotis N and Goldberg D E 1994 A niched Pareto genetic algorithm for multiobjective optimization *Proc. 1st IEEE Conf. on Evolutionary Computation, IEEE World Congr. on Computational Intelligence (Orlando, FL, 1994)* (Piscataway, NJ: IEEE) pp 82–7

Kursawe F 1991 A variant of evolution strategies for vector optimization *Parallel Problem Solving from Nature, 1st Workshop (Lecture Notes in Computer Science 496)* ed H-P Schwefel and R Männer (Berlin: Springer) pp 193–7

Richardson J T, Palmer M R, Liepins G and Hilliard M 1989 Some guidelines for genetic algorithms with penalty functions *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 191–7

Ritzel B J, Eheart J W and Ranjithan S 1994 Using genetic algorithms to solve a multiple objective groundwater pollution containment problem *Water Resources Res.* **30** 1589–603

Schaffer J D 1985 Multiple objective optimization with vector evaluated genetic algorithms *Genetic Algorithms and Their Applications: Proc. 1st Int. Conf. on Genetic Algorithms* ed J J Grefenstette (Hillsdale, NJ: Erlbaum) pp 93–100

Srinivas N and Deb K 1994 Multiobjective optimization using nondominated sorting in genetic algorithms *Evolutionary Comput.* **2** 221–48

Wienke D, Lucasius C and Kateman G 1992 Multicriteria target vector optimization of analytical procedures using a genetic algorithm Part I. Theory, numerical simulations and application to atomic emission spectroscopy *Anal. Chim. Acta* **265** 211–25

Wilson P B and Macleod M D 1993 Low implementation cost IIR digital filter design using genetic algorithms *IEE/IEEE Workshop on Natural Algorithms in Signal Processing (Chelmsford)* pp 4/1–8

C5.1 Introduction

Zbigniew Michalewicz

Abstract

This section provides a general introduction to constraint handling in evolutionary computation techniques.

In general, constraints are an integral part of the formulation of any problem. Dhar and Ranganathan (1990) wrote:

> ... Virtually all decision making situations involve constraints. What distinguishes various types of problems is the form of these constraints. Depending on how the problem is visualized, they can arise as rules, data dependencies, algebraic expressions, or other forms.
>
> Constraint satisfaction problems (CSPs) have been studied extensively in the operations research (OR) and artificial intelligence (AI) literature. In OR formulations constraints are quantitative, and the solver (such as the Simplex algorithm) optimizes (maximizes or minimizes) the value of a specified objective function subject to the constraints. In contrast, AI research has focused on inference-based approaches with mostly symbolic constraints. The inference mechanisms employed include theorem provers, production rule interpreters, and various labeling procedures such as those used in truth maintenance systems.

For example, in continuous domains, the general nonlinear programming problem is to find $\boldsymbol{x}$ so as to

$$\text{optimize } f(\boldsymbol{x}) \qquad \boldsymbol{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$$

where $\boldsymbol{x} \in \mathcal{F} \subseteq \mathcal{S}$. The set $\mathcal{S} \subseteq \mathbb{R}^n$ defines the search space and the set $\mathcal{F} \subseteq S$ defines a *feasible* search space. The search space $\mathcal{S}$ is defined as an n-dimensional rectangle in $\mathbb{R}^n$ (domains of variables defined by their lower and upper bounds):

$$l(i) \leq x_i \leq u(i) \qquad 1 \leq i \leq n$$

whereas the feasible set $\mathcal{F}$ is defined by an intersection of $\mathcal{S}$ and a set of additional $m \geq 0$ constraints:

$$g_j(\boldsymbol{x}) \leq 0 \quad \text{for } j = 1, \ldots, q \qquad h_j(\boldsymbol{x}) = 0 \quad \text{for } j = q+1, \ldots, m$$

(see Sections G9.1 and G9.9). G9.1, G9.9

In discrete domains, most problems are constrained: for example, the knapsack problem, set covering problem, vehicle routing problem, and all types of scheduling and timetabling problem are constrained.

In general, a search space $\mathcal{S}$ consists of two disjoint subsets of feasible and infeasible subspaces, $\mathcal{F}$ and $\mathcal{U}$, respectively (see figure C5.1.1). These subspaces need not be convex and they need not be connected (as, for example, in figure C5.1.1 where the feasible part $\mathcal{F}$ of the search space consists of four disjoined subsets). In solving optimization problems we search for a *feasible* optimum. During the search process we have to deal with various feasible and infeasible individuals; for example (see figure C5.1.2), at some stage of the evolution process, a population may contain some feasible (b, c, d, e, i, j, k, p) and infeasible individuals (a, f, g, h, l, m, n, o), while the optimum solution is marked by X.

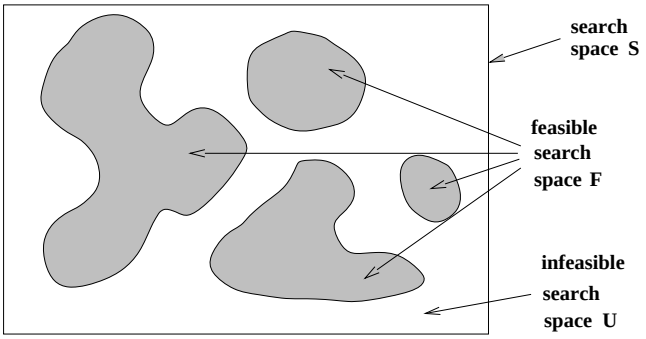


Figure C5.1.1. A search space and its feasible and infeasible parts.

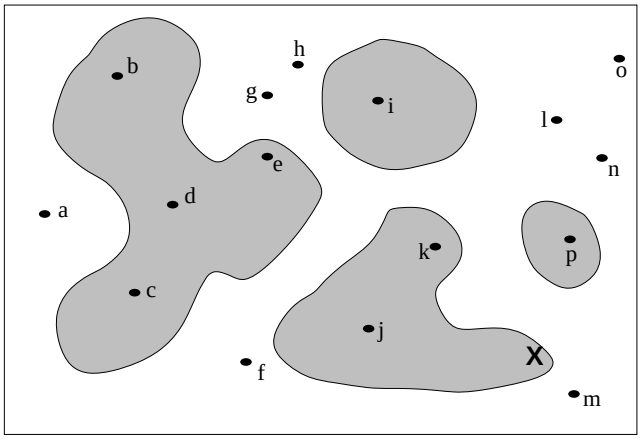


Figure C5.1.2. A population of 16 individuals, a–p.

The problem of how to deal with infeasible individuals is far from trivial. In general, we have to design two evaluation functions, eval_f and eval_u, for feasible and infeasible domains, respectively:

$$\text{eval}_f : \mathcal{F} \to \mathbb{R} \qquad \text{and} \qquad \text{eval}_u : \mathcal{U} \to \mathbb{R}.$$

There are many important questions to be addressed; these include:

- Should we choose to penalize infeasible individuals? In other words, should we extend the domain of function eval_f and assume that $\text{eval}_u(n) = \text{eval}_f(n) + \text{penalty}(n)$? (In the following discussion we continue referring to individuals just by plain letters of the alphabet, as displayed in figure C5.1.2; the reader should keep in mind that an individual, say, n, may represent a vector $\boldsymbol{x}$ in high-dimensional space). If so, how should such a penalty function penalty(n) be designed? In particular, should we consider infeasible individuals harmful and eliminate them from the population (Section C5.2)? C5.2
- Should we change the topology of the search space by using decoders, which interpret (transform) an individual into a feasible one (Section C5.3)? C5.3
- Should we 'repair' infeasible solutions by moving them into the closest point of the feasible space (e.g. the repaired version of m might be optimum X, figure C5.1.2)? In other words, should we assume that $\text{eval}_u(m) = \text{eval}_f(s)$, where s is a repaired version of m? If so, should we replace m by its repaired version s in the population or rather should we use a repair procedure for evaluation purposes only (Section C5.4)? C5.4
- Should we start with an initial population of feasible individuals and maintain the feasibility of offspring by using specialized operators (Section C5.5)? C5.5
- Should we use process solutions and constraints separately, or use cultural algorithms, or use co-evolutionary methods, that is, should we use some other, nonstandard constraint-handling technique (Section C5.6)? C5.6
- How should we locate feasible solutions (Section C5.7)? C5.7

The above questions are addressed in the following sections thus providing a detailed discussion on various aspects of constraint-handling techniques.

Reference

Dhar V and Ranganathan N 1990 Integer programming vs. expert systems: an experimental comparison *Commun. ACM* **33** 323–36

C5.2 Penalty functions

Alice E Smith and David W Coit

Abstract

This section begins with the motivation and general form of penalty functions as used in evolutionary computation. The main types of penalty function—constant, static, dynamic, and adaptive—are described within a common notation framework. References from the literature concerning these exterior penalty approaches are presented. The section concludes with a brief discussion of promising areas of future research in penalty methods for constrained optimization by evolutionary computation.

C5.2.1 Introduction to penalty functions

Penalty functions have been a part of the literature on constrained optimization for decades. Two basic types of penalty function exist: exterior penalty functions, which penalize infeasible solutions, and interior penalty functions, which penalize feasible solutions. It is the former type of penalty function which is discussed throughout this section; however the area of interior penalty functions is of potential research interest in evolutionary computation. The main idea of interior penalty functions is that an optimal solution requires that a constraint be active (i.e. tight) so that this optimal solution lies on the boundary between feasibility and infeasibility. Knowing this, a penalty is applied to feasible solutions when the constraint is not active (such solutions are called *interior solutions*). For a single constraint, this approach is straightforward (although it has not been seen in the evolutionary computation literature); however, for the more common case of multiple constraints, the implementation of interior penalty functions is considerably more complex.

Three degrees of exterior penalty functions exist: (i) barrier methods in which no infeasible solution is considered, (ii) partial penalty functions in which a penalty is applied near the feasibility boundary, and (iii) global penalty functions that are applied throughout the infeasible region (Schwefel 1995, p 16). In the area of combinatorial optimization, the popular Lagrangian relaxation method (Avriel 1976, Fisher 1981, Reeves 1993) is a variation on the same theme: temporarily relax the problem's most difficult constraints, using a modified objective function to avoid straying too far from the feasible region. In general, a penalty function approach is as follows. Given an optimization problem, the following is the most general formulation of constraints:

$$\min f(\boldsymbol{x}) \quad \text{such that } \boldsymbol{x} \in A, \boldsymbol{x} \in B \tag{C5.2.1}$$

where $\boldsymbol{x}$ is a vector of decision variables, the constraints '$\boldsymbol{x} \in A$' are relatively easy to satisfy, and the constraints '$\boldsymbol{x} \in B$' are relatively difficult to satisfy; the problem can be reformulated as

$$\min f(\boldsymbol{x} + p(d(\boldsymbol{x}, B)) \quad \text{such that } \boldsymbol{x} \in A \tag{C5.2.2}$$

where $d(\boldsymbol{x}, B)$ is a metric function describing the distance of the solution vector $\boldsymbol{x}$ from the region B, and $p(\cdot)$ is a monotonically nondecreasing penalty function such that $p(0) = 0$. If the exterior penalty function, $p(\cdot)$, grows quickly enough outside of B, the optimal solution of (C5.2.1) will also be optimal for (C5.2.2). Furthermore, any optimal solution of (C5.2.2) will (again, if $p(\cdot)$ grows quickly enough)

provide an upper bound on the optimum for (C5.2.1), and this bound will in general be tighter than that obtained by simply optimizing $f(\boldsymbol{x})$ over A.

In practice, the constraints '$\boldsymbol{x} \in B$' are expressed as inequality and equality constraints in the form of

$$g_i(\boldsymbol{x}) \leq 0 \qquad \text{for } i = 1, \ldots, q$$
$$h_i(\boldsymbol{x}) = 0 \qquad \text{for } i = q+1, \ldots, m$$

where

$$q = \text{number of inequality constraints}$$
$$m - q = \text{number of equality constraints.}$$

Various families of functions $p(\cdot)$ and $d(\cdot)$ have been studied for evolutionary optimization to dualize constraints. Different possible distance metrics, $d(\cdot)$, include a count of the number of violated constraints, the Euclidean distance between $\boldsymbol{x}$ and B as suggested by Richardson *et al* (1989), a linear sum of the individual constraint violations or a sum of the individual constraint violations raised to an exponent, κ. Variations of these approaches have been attempted with different degrees of success. Some of the more notable examples are described in the following sections.

It can be difficult to find a penalty function that is an effective and efficient surrogate for the missing constraints. The effort required to tune the penalty function to a given problem instance or repeatedly calculate it during search may negate any gains in eventual solution quality. As noted by Siedlecki and Sklansky (1989), much of the difficulty arises because the optimal solution will frequently lie on the boundary of the feasible region. Many of the solutions most similar to the genotype of the optimum solution will be infeasible. Therefore, restricting the search to only feasible solutions or imposing very severe penalties makes it difficult to find the schemata that will drive the population toward the optimum as shown in the research of Smith and Tate (1993), Anderson and Ferris (1994), Coit *et al* (1996), and Michalewicz (1995). Conversely, if the penalty is not severe enough, then too large a region is searched and much of the search time will be used to explore regions far from the feasible region. Then, the search will tend to stall outside the feasible region. A good comparison of six penalty function strategies applied to continuous optimization problems is given by Michalewicz (1995). These strategies include both static and dynamic approaches, as discussed below, as well as some less generic approaches such as sequential constraint handling (Schoenauer and Xanthakis 1993) and forcing all infeasible solutions to be dominated by all feasible solutions in a given generation (Powell and Skolnick 1993).

C5.2.2 Static penalty functions

A simple method to penalize infeasible solutions is to apply a constant penalty to those solutions that violate feasibility in any way. The penalized objective function would then be the unpenalized objective function plus a penalty (for a minimization problem). A variation is to construct this simple penalty function as a function of the number of constraints violated, where there are multiple constraints. The penalty function for a problem with m constraints would then be as below (for a minimization problem):

$$f_{\mathrm{p}}(\boldsymbol{x}) = f(\boldsymbol{x}) + \sum_{i=1}^{m} C_i \delta_i \qquad \text{where } \begin{cases} \delta_i = 1 & \text{if constraint } i \text{ is violated} \\ \delta_i = 0 & \text{if constraint } i \text{ is satisfied.} \end{cases} \tag{C5.2.3}$$

$f_{\mathrm{p}}(\boldsymbol{x})$ is the penalized objective function, $f(\boldsymbol{x})$ is the unpenalized objective function, and C_i is a constant imposed for violation of constraint i. This penalty function is based only on the number of constraints violated, and is generally inferior to the second approach, based on some distance metric from the feasible region (Goldberg 1989, Richardson *et al* 1989).

More common and more effective is to penalize according to distance to feasibility, or the 'cost to completion', as termed by Richardson *et al* (1989). This was done crudely in the constant penalty functions of the preceding paragraph by assuming distance can be stated solely by number of constraints violated. A more sophisticated and more effective penalty includes a distance metric for each constraint, and adds a penalty that becomes more severe with distance from feasibility. Complicating this approach is the assumption that the distance metric chosen appropriately provides information concerning the nearness

of the solution to feasibility, and the further implicit assumption that this nearness to feasibility is relevant in the same magnitude to the fitness of the solution. Distance metrics can be continuous (see, for example, Juliff 1993) or discrete (see, for example, Patton *et al* 1995), and could be linear or nonlinear (see, for example, Le Riche *et al* 1995).

A general formulation is as follows for a minimization problem:

$$f_{\mathrm{p}}(\boldsymbol{x}) = f(\boldsymbol{x}) + \sum_{i=1}^{m} C_i d_i^{\kappa} \quad \text{where } d_i = \begin{cases} \delta_i g_i(\boldsymbol{x}) & \text{for } i = 1, \ldots, q \\ |h_i(\boldsymbol{x})| & \text{for } i = q+1, \ldots, m. \end{cases} \tag{C5.2.4}$$

d_i is the distance metric of constraint i applied to solution $\boldsymbol{x}$ and κ is a user-defined exponent, with values of κ of one or two often used. Constraints 1–q are inequality constraints, so the penalty will only be activated when the constraint is violated (as shown by the δ function above), while constraints $(q+1)$–m are equality constraints which will activate the penalty if there is any distance between the solution value and the constraint value (as shown in the absolute distance above). In equation (C5.2.4) above, defining C_i is more difficult. The advice from Richardson *et al* (1989) is to base C_i on the expected or maximum cost to repair the solution (i.e. alter the solution so it is feasible). For most problems, however, it is not possible to determine C_i using this rationale. Instead, it must be estimated based on the relative scaling of the distance metrics of multiple constraints, the difficulty of satisfying a constraint, and the seriousness of a constraint violation, or be determined experimentally.

Many researchers in evolutionary computation have explored variations of distance-based static penalty functions (e.g. Baeck and Khuri 1994, Goldberg 1989, Huang *et al* 1994, Olsen 1994, Richardson *et al* 1989). One example (Thangiah 1995) uses a linear combination of three constant distance-based penalties for the three constraints of the vehicle routing with time windows problem. Another novel example is from Le Riche *et al* (1995) where two separate distance-based penalty functions are used for each constraint in two genetic algorithm segregated subpopulations. This 'double penalty' somewhat improved robustness to penalty function parameters since the feasible optimum is approached with both a severe and a lenient penalty. Homaifar *et al* (1994) developed a unique static penalty function with multiple violation levels established for each constraint. Each interval is defined by the relative degree of constraint violation. For each interval l, a unique constant, C_{il}, is then used as a penalty function coefficient. This approach has the considerable disadvantage of requiring iterative tuning through experimentation of a large number of parameters.

C5.2.3 Dynamic penalty functions

The primary deficiency with static penalty functions is the inability of the user to determine criteria for the C_i coefficients. Also, there are conflicting objectives involved with allowing exploration of the infeasible region, yet still requiring that the final solution be feasible. A variation of distance-based penalty functions that alleviates many of these difficulties is to incorporate a dynamic aspect that (generally) increases the severity of the penalty for a given distance as the search progresses. This has the property of allowing highly infeasible solutions early in the search, while continually increasing the penalty imposed to eventually move the final solution to the feasible region. A general form of a distance-based penalty method incorporating a dynamic aspect based on length of search, t, is as follows for a minimization problem:

$$f_{\mathrm{p}}(\boldsymbol{x}, t) = f(\boldsymbol{x}) + \sum_{i=1}^{m} s_i(t) d_i^{\kappa} \tag{C5.2.5}$$

where $s_i(t)$ is a function monotonically nondecreasing in value with t. Metrics for t include number of generations or the number of solutions searched. Recent uses of this approach include the work of Joines and Houck (1994) for continuous function optimization and Olsen (1994) and Michalewicz and Attia (1994), which compare several penalty functions, all of which consider distance, but some also consider evolution time. A common objective of these dynamic penalty formulations is that they result in feasible solutions at the end of evolution. If $s_i(t)$ is too lenient, final infeasible solutions may result, and if $s_i(t)$ is too severe, the search may converge to nonoptimal feasible solutions. Therefore, these penalty functions typically require problem-specific tuning to perform well. One explicit example of $s_i(t)$ is as follows, from Joines and Houck (1994):

$$s_i(t) = (C_i t)^{\alpha}$$

where α is constant equal to one or two, as defined by Joines and Houck.

C5.2.4 Adaptive penalty functions

While incorporating distance together with the length of the search into the penalty function has been generally effective, these penalties ignore any other aspects of the search. In this respect, they are not adaptive to the ongoing success (or lack thereof) of the search and cannot guide the search to particularly attractive regions or away from unattractive regions based on what has already been observed. A few authors have proposed making use of such search-specific information. Siedlecki and Sklansky (1989) discuss the possibility of adaptive penalty functions, but their method is restricted to *binary-string* encodings C1.2 with a single constraint, and involves considerable computational overhead.

Bean and Hadj-Alouane (1992) and Hadj-Alouane and Bean (1992) propose penalty functions that are revised based on the feasibility or infeasibility of the best, penalized solution during recent generations. Their penalty function allows either an increase or a decrease of the imposed penalty during evolution as shown below, and was demonstrated on multiple-choice integer programming problems with one constraint. This involves the selection of two constants, β_1 and β_2 ($\beta_1 > \beta_2 > 1$), to adaptively update the penalty function multiplier, and the evaluation of the feasibility of the best solution over successive intervals of N_f generations. As the search progresses, the penalty function multiplier is updated every N_f generations based on whether or not the best solution was feasible during that interval. Specifically, the penalty function is as follows;

$$f_p(\boldsymbol{x}, k) = f(\boldsymbol{x}) + \sum_{i=1}^{m} \lambda_k d_i^{\kappa}$$
$$\lambda_{k+1} = \begin{cases} \lambda_k \beta_1 & \text{if previous } N_f \text{ generations have only infeasible best solution} \\ \lambda_k / \beta_2 & \text{if previous } N_f \text{ generations have only feasible best solution} \\ \lambda_k & \text{otherwise.} \end{cases} \tag{C5.2.6}$$

Smith and Tate (1993) and Tate and Smith (1995) used both search length and constraint severity feedback in their penalty function, which was enhanced by the work of Coit *et al* (1996). This penalty function involves the estimation of a near-feasible threshold (NFT) for each constraint. Conceptually, the NFT is the threshold distance from the feasible region at which the user would consider the search as 'getting warm'. The penalty function encourages the evolutionary algorithm to explore within the feasible region and the NFT neighborhood of the feasible region, and discourage search beyond that threshold. This formulation is given below:

$$f_p(\boldsymbol{x}, t) = f(\boldsymbol{x}) + (F_{\text{feas}}(t) - F_{\text{all}}(t)) \sum_{i=1}^{m} \left(\frac{d_i}{\text{NFT}_i} \right)^{\kappa} \tag{C5.2.7}$$

where $F_{\text{all}}(t)$ denotes the unpenalized value of the best solution yet found, and $F_{\text{feas}}(t)$ denotes the value of the best feasible solution yet found. The $F_{\text{all}}(t)$ and $F_{\text{feas}}(t)$ terms serve several purposes. First, they provide adaptive scaling of the penalty based on the results of the search. Second, they combine with the NFT_i term to provide a search-specific and constraint-specific penalty.

The general form of NFT_i is

$$\text{NFT}_i = \frac{\text{NFT}_{0i}}{1 + \Lambda_i} \tag{C5.2.8}$$

where NFT_{0i} is an upper bound for NFT_i. Λ_i is a dynamic search parameter used to adjust NFT_i based on the search history. In the simplest case, Λ_i can be set to zero and a static NFT_i results. Λ_i can also be defined as a function of the search, for example, a function of the generation number (t), i.e. $\Lambda_i = f(t) = \lambda_i t$. A positive value of λ_i results in a monotonically decreasing NFT_i (and, thus, a larger penalty) and a larger λ_i more quickly decreases NFT_i as the search progresses, incorporating both adaptive and dynamic elements.

If NFT_i is intuitively ill defined, it can be set at a large value initially with a positive constant λ_i used to iteratively guide the search to the feasible region. This dynamic NFT_i circumvents the need to perform experimentation to determine appropriate penalty function parameter values. However, if problem-specific information is at hand, a more efficient search can take place by *a priori* defining a tighter region or even static values of NFT_i.

C5.2.5 Future directions in penalty functions

Two areas requiring further research are the development of completely adaptive penalty functions that require no user-specified constants and the development of improved adaptive operators to exploit characteristics of the search as they are found. The notion of adaptiveness is to leverage the information gained during evolution to improve both the effectiveness and the efficiency of the penalty function used. Another area of interest is to explore the assumption that multiple constraints can be linearly combined to yield an appropriate penalty function. This implicit assumption of all penalty functions used in the literature assumes that constraint violations incur independent penalties and therefore there is no interaction between constraints. Intuitively, this seems to be a possibly erroneous assumption, and one could make a case for a penalty that increases more than linearly with the number of constraints violated.

References

Anderson E J and Ferris M C 1994 Genetic algorithms for combinatorial optimization: the assembly line balancing problem *ORSA J. Comput.* **6** 161–73

Avriel M 1976 *Nonlinear Programming: Analysis and Methods* (Englewood Cliffs, NJ: Prentice-Hall)

Baeck T and Khuri S 1994 An evolutionary heuristic for the maximum independent set problem *Proc. 1st IEEE Conf. on Evolutionary Compution (Orlando, FL, June 1994)* (Piscataway, NJ: IEEE) pp 531–5

Bean J C and Hadj-Alouane A B 1992 *A Dual Genetic Algorithm for Bounded Integer Programs* University of Michigan Technical Report 92-53; Revue francaise d'automatique, d'informatique et de recherche operationnelle: Recherche operationnelle, at press (in French)

Coit D W, Smith A E and Tate D M 1996 Adaptive penalty methods for genetic optimization of constrained combinatorial problems *INFORMS J. Comput.* **8** 173–82

Fisher M L 1981 The Lagrangian relaxation method for solving integer programming problems *Management Sci.* **27** 1–18

Goldberg D E 1989 *Genetic Algorithms in Search Optimization and Machine Learning* (Reading, MA: Addison-Wesley)

Hadj-Alouane A B and Bean J C 1992 *A Genetic Algorithm for the Multiple-Choice Integer Program* University of Michigan Technical Report 92-50; *Operations Res.* at press

Homaifar A, Lai S H-Y and Qi Z 1994 Constrained optimization via genetic algorithms *Simulation* **62** 242–54

Huang W-C, Kao C-Y and Horng J-T 1994 A genetic algorithm approach for set covering problem *Proc. 1st IEEE Conf. on Evolutionary Computation (Orlando, FL, June 1994)* (Piscataway, NJ: IEEE) pp 569–73

Joines J A and Houck C R 1994 On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with GA's *Proc. 1st IEEE Conf. on Evolutionary Computation (Orlando, FL, 1994)* (Piscataway, NJ: IEEE) pp 579–84

Juliff K 1993 A multi-chromosome genetic algorithm for pallet loading *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 467–73

Le Riche R G, Knopf-Lenoir C and Haftka R T 1995 A segregated genetic algorithm for constrained structural optimization *Proc. 6th Int. Conf. on Genetic Algorithms (Pittsburgh, PA, July 1995)* ed L J Eshelman (San Mateo, CA: Morgan Kaufmann) pp 558–65

Michalewicz Z 1995 Genetic algorithms numerical optimization and constraints *Proc. 6th Int. Conf. on Genetic Algorithms (Pittsburgh, PA, July 1995)* ed L J Eshelman (San Mateo, CA: Morgan Kaufmann) pp 151–8

Michalewicz Z and Attia N 1994 Evolutionary optimization of constrained problems *Proc. 3rd Ann. Conf. on Evolutionary Programming (San Diego, CA, February 1994)* ed A V Sebald and L J Fogel (Singapore: World Scientific) pp 98–108

Olsen A L 1994 Penalty functions and the knapsack problem *Proc. 1st IEEE Conf. on Evolutionary Computation (Orlando, FL, June 1994)* (Piscataway, NJ: IEEE) pp 554–8

Patton A L, Punch W F III and Goodman E D 1995 A standard GA approach to native protein conformation prediction *Proc. 6th Int. Conf. on Genetic Algorithms (Pittsburgh, PA, July 1995)* ed L J Eshelman (San Mateo, CA: Morgan Kaufmann) pp 574–81

Powell D and Skolnick M M 1993 Using genetic algorithms in engineering design optimization with non-linear constraints *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 424–30

Reeves C R 1993 *Modern Heuristic Techniques for Combinatorial Problems* (New York: Wiley)

Richardson J T, Palmer M R, Liepins G and Hilliard M 1989 Some guidelines for genetic algorithms with penalty functions *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, June 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 191–7

Schoenauer M and Xanthakis S 1993 Constrained GA optimization *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 573–80

Schwefel H-P 1995 *Evolution and Optimum Seeking* (New York: Wiley)

Siedlecki W and Sklansky J 1989 Constrained genetic optimization via dynamic reward–penalty balancing and its use in pattern recognition *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, June 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 141–50

Smith A E and Tate D M 1993 Genetic optimization using a penalty function *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 499–505

Tate D M and Smith A E 1995 Unequal area facility layout using genetic search *IIE Trans.* **27** 465–72

Thangiah S R 1995 An adaptive clustering method using a geometric shape for vehicle routing problems with time windows *Proc. 6th Int. Conf. on Genetic Algorithms (Pittsburgh, PA, July 1995)* ed L J Eshelman (San Mateo, CA: Morgan Kaufmann) pp 536–43

C5.3 Decoders

Zbigniew Michalewicz

Abstract

This section discusses one particular approach for constraint-handling, namely, a use of decoders. Decoders process (or interpret) instructions incorporated in the chromosomal material of an individual; this activity results in the construction of a feasible solution.

C5.3.1 Introduction

Decoders offer an interesting option for all practitioners of evolutionary techniques. In these techniques a chromosome 'gives instructions' to a decoder or 'is interpreted' by a decoder on how to build a feasible solution. For example, a sequence of items for the knapsack problem can be interpreted as 'take an item if possible'—such interpretation would lead always to feasible solutions. Let us consider the following scenario: we try to solve the 0–1 *knapsack problem* with n items; the profit and weight of the ith item are p_i and w_i, respectively. We can sort all items in decreasing order of p_i/w_i values and interpret the binary string C5.4, G9.7

$$(1100110001001110101001010111010101 \ldots 0010)$$

in the following way. Take the first item from the list (i.e. the item with the largest ratio of profit per weight) if the item fits in the knapsack. Continue with the second, fifth, sixth, tenth, and so on, items from the sorted list (i.e. continue with items with corresponding ones in the binary string), until the knapsack is full or there are no more items available (note that the binary string of all ones corresponds to a greedy solution). Any sequence of bits would translate into a feasible solution; any feasible solution may have many possible codes (which may differ in the rightmost string part). We can apply classical binary operators (crossover and mutation): any offspring is clearly feasible.

C5.3.2 The traveling salesman problem

A similar approach has been tried for solving the *traveling salesman problem* (Grefenstette *et al* 1985). For example, a chromosome may represent a tour as a list of n cities; the ith element of the list is a number in the range from 1 to $n - i + 1$. The idea behind such decoder is as follows. There is some ordered list of cities C, which serves as a reference point for lists in ordinal representations (like a sorted sequence of items for the knapsack problem discussed earlier). Assume, for example, that such an ordered list (a reference point) is simply G9.5

$$C = (1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9).$$

A tour

$$1\text{–}2\text{–}4\text{–}3\text{–}8\text{–}5\text{–}9\text{–}6\text{–}7$$

is then represented as a list l of references,

$$l = (1\ 1\ 2\ 1\ 4\ 1\ 3\ 1\ 1)$$

and should be interpreted as follows: the first number on the list l is 1, so take the first city from the list C as the first city of the tour (city number 1), and remove it from C. At this stage the partial tour is (1). The next number on the list l is also 1, so take the first city from the current list C as the next city of the tour (city number 2), and remove it from C. At this stage the partial tour is (1, 2), and so on. The main advantage of the ordinal representation is that the classical crossover works: any two tours in the ordinal representation, cut after some position and crossed together, would produce two offspring, each of them being a legal tour. For example, the two parents

$$p_1 = (1\ 1\ 2\ 1 \mid 4\ 1\ 3\ 1\ 1)$$

and

$$p_2 = (5\ 1\ 5\ 5 \mid 5\ 3\ 3\ 2\ 1)$$

which correspond to the tours

1 –2–4–3–8–5–9–6–7

and

5–1–7–8–9–4–6–3–2

with the crossover point marked by |, would produce the following offspring:

$$o_1 = (1\ 1\ 2\ 1\ 5\ 3\ 3\ 2\ 1)$$

and

$$o_2 = (5\ 1\ 5\ 5\ 4\ 1\ 3\ 1\ 1).$$

These offspring correspond to

1–2–4–3–9–7–8–6–5

and

5–1–7–8–6–2–9–3–4.

There are many other examples of how decoders have been used for a particular application. These include work on *scheduling problems* (see, for example, Bagchi *et al* 1991 and Syswerda 1991), *pallet loading* (Juliff 1993) and partitioning (Jones and Beltramo 1991). F1.5, F1.7

C5.3.3 Formal description

More formally, a decoder is a mapping T from a representation space (e.g. a space of binary strings, vectors of integer numbers and the like) into a feasible part of the solution space $\mathcal{F}$—viewing decoders from this perspective, evolutionary computation technique with a decoder is identical to so-called morphogenic evolutionary techniques (Angeline 1995), which include mappings (i.e. development functions) between representations that evolve (i.e. evolved representations) and representations that constitute the input for the evaluation function (i.e. evaluated representation). A graphical example of such a mapping is given in figure C5.3.1, where the mapping T transforms a point d in the representation space (figure C5.3.1(b)) into a feasible solution s (figure C5.3.1(a)).

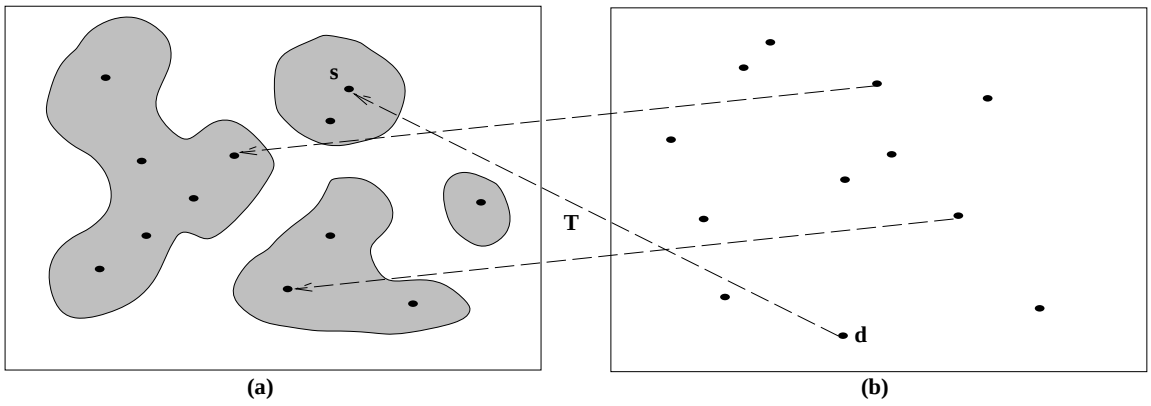


Figure C5.3.1. Transformation T between solutions in (a) original and (b) decoder's space.

However, it is important that several conditions are satisfied (Palmer and Kershenbaum 1994):

- for each solution $s \in \mathcal{F}$ there is a solution d from the representation space
- each solution d from the representation space corresponds to a feasible solution $s \in \mathcal{F}$
- all solutions in $\mathcal{F}$ should be represented by the same number of solutions (codings) d.

Additionally, it is reasonable to request that:

- the transformation T is computationally fast
- it has a locality feature in the sense that small changes in a solution from the representation space result in small changes in the (feasible) solution itself.

Also, as stated by Davis (1987),

> If one builds a 'decoder' into the evaluation procedure that intelligently avoids building an illegal individual from the chromosome, the result is frequently computation-intensive to run. Further, not all constraints can be easily implemented in this way.

References

Angeline P J 1995 Morphogenic evolutionary computation: introduction, issues, and examples *Proc. 4th Ann. Conf. on Evolutionary Programming (San Diego, CA, March 1995)* ed J R McDonnell, R G Reynolds and D B Fogel (Cambridge, MA: MIT Press) pp 387–401

Bagchi S, Uckun S, Miyabe Y and Kawamura K 1991 Exploring problem-specific recombination operators for job shop scheduling *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, July 1991)* ed R Belew and L Booker (San Mateo, CA: Morgan Kaufmann) pp 10–7

Davis L (ed) 1987 *Genetic Algorithms and Simulated Annealing* (Los Altos, CA: Morgan Kaufmann)

Grefenstette J J, Gopal R, Rosmaita B and Van Gucht D 1985 Genetic algorithm for the TSP *Proc. 1st Int. Conf. on Genetic Algorithms (Pittsburgh, PA, July 1985)* ed J J Grefenstette (San Mateo, CA: Morgan Kaufmann) pp 160–8

Jones D R and Beltramo M A 1991 Solving partitioning problems with genetic algorithms *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, July 1991)* ed R Belew and L Booker (San Mateo, CA: Morgan Kaufmann) pp 442–9

Juliff K 1993 A multi-chromosome genetic algorithm for pallet loading *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 467–73

Palmer C C and Kershenbaum A 1994 Representing trees in genetic algorithms *Proc. IEEE Int. Conf. on Evolutionary Computation (Orlando, FL, June–July 1994)* pp 379–84

Syswerda G 1991 Schedule optimization using genetic algorithms *Handbook of Genetic Algorithms* ed L Davis (New York: Van Nostrand Reinhold) pp 332–49

C5.4 Repair algorithms

Zbigniew Michalewicz

Abstract

This section discusses one particular approach for constraint handling, namely, a use of repair algorithms. These algorithms map any infeasible individual into a feasible one. Such repairs can be made for evaluation purposes only, or the repaired individual can replace the original one in the population.

C5.4.1 Introduction

Repair algorithms enjoy a particular popularity in the evolutionary computation community: for many combinatorial optimization problems (e.g. the traveling salesman problem, the knapsack problem, and the set covering problem) it is relatively easy to *repair* an infeasible individual. Such a repaired version can be used either for evaluation only; that is,

$$\text{eval}_u(y) = \text{eval}_f(x)$$

where x is a repaired (i.e. feasible) version of y, or it can also replace (with some probability) the original individual in the population. Note that the repaired version of solution m (figure C5.1.2) might be the optimum X. C5.1

The process of repairing infeasible individuals is related to combination of learning and evolution (the so-called *Baldwin effect*, Whitley *et al* 1994). Learning (as local search in general, and local search for the closest feasible solution, in particular) and evolution interact with each other: the evaluation of the improvement (again, improvement in the sense of finding a repaired, feasible solution) is transferred to the individual. In this way a local search is analogous to learning that occurs during one generation of a particular string. Note that the repair process is used *only* for evaluation of an individual; the repaired version of the individual does not replace the original one. C3.4.1

The weakness of these methods is in their problem dependence. For each particular problem a specific repair algorithm should be designed. Moreover, there are no standard heuristics on design of such algorithms; usually it is possible to use a greedy repair or random repair or incorporate any other heuristic which would guide the repair process. Also, for some problems the process of repairing infeasible individuals may be as complex as solving the original problem. This is the case for the nonlinear transportation problem (see Michalewicz 1993), most scheduling and timetable problems, and many others.

The question of replacing repaired individuals is related to so-called *Lamarckian evolution* (Whitley *et al* 1994), which assumes that an individual improves during its lifetime and that the resulting improvements are coded back into the chromosome. As stated by Whitley *et al* (1994), A2.1

> Our analytical and empirical results indicate that Lamarckian strategies are often an extremely fast form of search. However, functions exist where both the simple genetic algorithm without learning and the Lamarckian strategy used [...] converge to local optima while the simple genetic algorithm exploiting the Baldwin effect converges to a global optimum.

This is why it is necessary to use the replacement strategy very carefully.

Recently Orvosh and Davis (1993) reported a so-called 5% rule: this heuristic rule states that in many combinatorial optimization problems an evolutionary computation technique with a repair algorithm provides the best results when 5% of repaired individuals replace their infeasible originals. However, many recent experiments (see e.g. Michalewicz 1996) have indicated that for many combinatorial optimization problems this rule does not apply. Either a different percentage gives better results, or there is no significant difference in the performance of the algorithm for various probabilities of replacement.

It seems that the 'optimal' probability of replacement is problem dependent and it may change over the evolution process as well. Further research is required to compare different heuristics for setting this parameter, which is of great importance for all repair-based methods.

We shall illustrate the above points on two examples (taken from discrete and continuous domains): the 0–1 knapsack problem and the nonlinear programming problem.

The 0–1 knapsack problem can be formulated as follows: for a given set of weights w_i, profits p_i, and capacity C, find a binary vector $\boldsymbol{x} = \langle x_1, \ldots, x_n \rangle$, such that

$$\sum_{i=1}^{n} x_i w_i \leq C$$

and for which

$$\mathcal{P}(\boldsymbol{x}) = \sum_{i=1}^{n} x_i p_i$$

is maximum.

A binary string of the length n represents a solution $\boldsymbol{x}$ to the problem: the ith item is selected for the knapsack iff $x[i] = 1$. The evaluation of each string is determined on the basis of its feasibility; the evaluation measure eval_f for a feasible string $\boldsymbol{x}$ is

$$\text{eval}_\text{f}(\boldsymbol{x}) = \sum_{i=1}^{n} x_i p_i$$

whereas the evaluation measure eval_u for a infeasible string $\boldsymbol{x}$ is

$$\text{eval}_\text{u}(\boldsymbol{x}) = \sum_{i=1}^{n} x'_i p_i$$

where vector $\boldsymbol{x}'$ is a repaired version of the original vector $\boldsymbol{x}$.

The procedure for converting infeasible $\boldsymbol{x}$ into feasible $\boldsymbol{x}'$ is straightforward:

Input: $\boldsymbol{x}$

Output: $\boldsymbol{x}'$, the repaired version of $\boldsymbol{x}$

knapsack-overfilled $\leftarrow$ false;
$\boldsymbol{x}' \leftarrow \boldsymbol{x}$
if $\sum_{i=1}^{n} x'_i w_i > C$
then
    knapsack-overfilled $\leftarrow$ true
fi
while (knapsack-overfilled) **do**
    $i \leftarrow$ **select** an item from the knapsack
    remove the selected item from the knapsack:
        $x'_i \leftarrow 0$;
    if $\sum_{i=1}^{n} x'_i w_i \leq C$
    then
        knapsack-overfilled $\leftarrow$ false
    fi
od

C5.4.2 First example

There are still several possible repair methods which follow the outline of this repair procedure; they may differ in selection procedure **select**, which chooses an item for removal from the knapsack. For example, the procedure **select** (i) may select a random element from the knapsack, (ii) may select the first available element from the left (right) of the list, (iii) may sort all items in the knapsack in decreasing order of their profit to weight ratios and always choose the last item (from the list of available items) for deletion (i.e. greedy repair), or (iv) may sort all items in the knapsack in decreasing order of their profit to weight ratios and choose an item (from the list of available items) for deletion with respect to some probability distribution (items with a larger ratio would have smaller probability of selection). Other repair methods are also possible.

In general, there are two categories of repair methods. Some of them (such as (i) and (iv) in the previous paragraph) contain an element of randomness; consequently, it is possible that two identical solutions have different evaluation measures. On the other hand, other repair methods (such as (ii) or (iii) in the previous paragraph) are deterministic.

From experiments reported by Michalewicz (1996) it seems that deterministic (greedy) repair gives much better results than random repairs. Additionally, the experiments did not confirm the 5% replacement rule: either a different percentage gave better results, or there was no significant difference in the performance of the algorithm for various probabilities of replacement. In most cases, the higher the replacement ratio, the better the result (as a rule of a thumb, experiments with the 0–1 knapsack problem suggest a replacement ratio of 1.0!).

C5.4.3 Second example

The second example of a repair process in evolutionary techniques is taken from a continuous domain: it is the nonlinear programming problem. The problem is formulated as follows: find $\boldsymbol{x}$ so as to

$$\text{optimize } f(\boldsymbol{x}) \qquad \boldsymbol{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$$

subject to

$$g_j(\boldsymbol{x}) \leq 0 \quad \text{for } j = 1, \ldots, q \qquad \text{and} \qquad h_j(\boldsymbol{x}) = 0 \quad \text{for } j = q+1, \ldots, m.$$

Michalewicz and Nazhiyath (1995) reported on implementation of a new system, Genocop III (see Section G9.9.3 for a full description of the system). Genocop III maintains two separate populations, G9.9.3 where a development in one population influences evaluations of individuals in the other population. The first population P_s consists of so-called search points. Search points need not be feasible; the variables just stay within specified limits, that is, they satisfy domain constraints. (In Genocop III it is also possible to define linear constraints as a separate set of constraints, which are handled by specialized operators; we refer the reader to Section G9.9 for more details of this option.) The second population P_r consists of G9.9 so-called reference points; these points are fully feasible, that is, they satisfy *all* constraints (if the system cannot find any reference point, the user is prompted for it).

Reference points $\boldsymbol{r}$ from P_r, being feasible, are evaluated directly by the objective function (i.e. $\text{eval}_f(\boldsymbol{r}) = f(\boldsymbol{r})$). On the other hand, search points from P_s are repaired for evaluation and the repair process for $\boldsymbol{s} \in P_s$ works as follows. If $\boldsymbol{s}$ is feasible, then $\text{eval}_f(\boldsymbol{s}) = f(\boldsymbol{s})$. Otherwise (i.e. $\boldsymbol{s} \notin \mathcal{F}$), the system selects one of the reference points, for example, $\boldsymbol{r}$ from P_r and creates a sequence of points $\boldsymbol{z}_i$ from a segment between $\boldsymbol{s}$ and $\boldsymbol{r}$: $\boldsymbol{z}_i = a_i \boldsymbol{s} + (1 - a_i)\boldsymbol{r}$. This can be done either (i) in a random way by generating random numbers a_i from the range $\langle 0, 1\rangle$, or (ii) in a deterministic way by setting $a_i = 1/2, 1/4, 1/8, \ldots$ until a feasible point is found. Figure C5.4.1 illustrates the point.

The system has a few additional parameters. As explained in the previous paragraph, two repair methods are available: random or deterministic. Also, it is possible to specify the way a reference point is selected for a repair process. This selection is random either with a uniform probability distribution (all reference points have equal chances for selection) or with a probability distribution of reference points that depends on their evaluations (a ranking method is used). Clearly, in different generations the same search point $\mathcal{S}$ can evaluate to different values due to the random nature of the repair process.

Additionally, if $f(\boldsymbol{z})$ is better than $f(\boldsymbol{r})$, then the point $\boldsymbol{z}$ replaces $\boldsymbol{r}$ as a new reference point in the population of reference points P_r. Also, $\boldsymbol{z}$ replaces $\boldsymbol{s}$ in the population of search points P_s with some probability of replacement p_r.

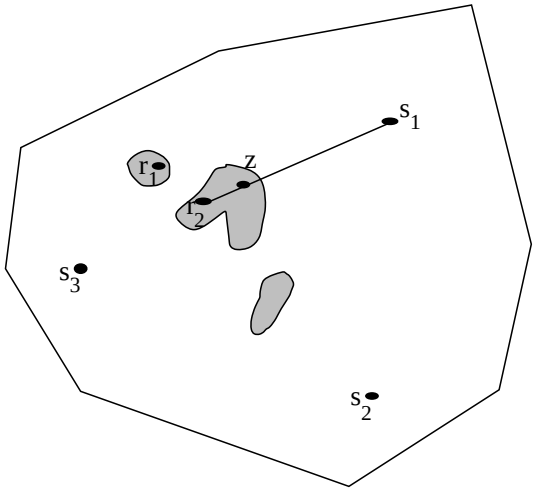


Figure C5.4.1. The repair process. A solution (search point) s_1 is repaired (point z) with respect to the reference solution r_2. The feasible areas of the search space are shaded.

It was interesting to check whether some p% rule (like the 5% rule of Orvosh and Davis, 1993, reported for discrete domains) would emerge from experiments for numerical optimization problems. For this purpose one of the test problems for Genocop III was selected.

The problem (Keane 1994) is to maximize a function:

$$f(\boldsymbol{x}) = \left| \frac{\sum_{i=1}^{n} \cos^4(x_i) - 2\prod_{i=1}^{n} \cos^2(x_i)}{\left(\sum_{i=1}^{n} \mathrm{i}x_i^2\right)^{1/2}} \right|$$

where

$$\prod_{i=1}^{n} x_i > 0.75 \qquad \sum_{i=1}^{n} x_i < 7.5n \qquad 0 < x_i < 10 \quad \text{for } 1 \le i \le n.$$

Genocop III was run for the case of $n = 20$ for 10 000 generations with different values of replacement ratio. It was interesting to note the increase of the performance (in terms of the best solution found) of the system when the replacement ratio was increased gradually from 0.00 to 0.15%. For the ratio of 0.15% the best solution found was

$$\begin{aligned}\boldsymbol{x} = (&3.163\,113\,59, 3.131\,504\,30, 3.095\,158\,58, 3.060\,165\,88, 3.031\,035\,66,\\ &2.991\,585\,49, 2.958\,025\,93, 2.922\,858\,95, 0.486\,843\,88, 0.477\,322\,79,\\ &0.480\,444\,73, 0.487\,909\,11, 0.484\,504\,37, 0.448\,070\,32, 0.468\,777\,60,\\ &0.456\,485\,06, 0.447\,626\,08, 0.449\,139\,86, 0.443\,908\,63, 0.451\,493\,32)\end{aligned}$$

where $f(\boldsymbol{x}) = 0.803\,510\,67$. However, further increases deteriorated the performance of the system; often the system converged to points $\boldsymbol{y} \in \mathcal{F}$, where $0.75 \le f(\boldsymbol{y}) \le 0.78$.

It is too early to claim a 15% replacement rule for continuous domains; however, $p_r = 0.15$ gave also the best results for other *test cases*. G9.9.4

C5.4.4 Conclusion

Clearly, further research is necessary to investigate the relationship between optimization problems and repair techniques (these include repair methods as well as replacement rates).

References

Keane A 1994 *Genetic Algorithms Digest* **8** issue 16

Michalewicz Z 1993 A hierarchy of evolution programs: an experimental study *Evolutionary Comput.* **1** 51–76

——1996 *Genetic Algorithms + Data Structures = Evolution Programs* 3rd edn (New York: Springer)

Michalewicz Z and Nazhiyath G 1995 Genocop III: a co-evolutionary algorithm for numerical optimization problems with nonlinear constraints *Proc. 2nd IEEE Int. Conf. on Evolutionary Computation (Perth, 1995)* pp 647–51

Orvosh D and Davis L 1993 Shall we repair? Genetic algorithms, combinatorial optimization, and feasibility constraints *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) p 650

Whitley D, Gordon V S and Mathias K 1994 Lamarckian evolution, the Baldwin effect and function optimization *Proc. 3rd Conf. on Parallel Problem Solving from Nature (October 1994, Jerusalem)* ed Yu Davidor, H-P Schwefel and R Männer (Berlin: Springer) pp 6–15

C5.5 Constraint-preserving operators

Zbigniew Michalewicz

Abstract

This section discusses one particular approach for constraint handling, namely, a use of domain-specific genetic operators which preserve feasibility of solutions.

C5.5.1 Introduction

Many researchers have successfully experimented with specialized operators which preserve feasibility of individuals. These specialized operators incorporate problem-specific knowledge; the purpose of incorporating domain-based heuristics into operators (Surry *et al* 1995) is

> . . . to build and use genetic operators that "understand" the constraints, in the sense that they never produce infeasible solutions (ones that violate the constraints). [. . .] The search is thus reformulated as an unconstrained optimization problem over the reduced space.

The main disadvantages connected with this approach are that (i) the problem-specific operators must be tailored for a particular application, and that (ii) it is very difficult to provide any formal analysis of such a system (however, important work towards understanding the way in which operators manipulate chromosomes is reported by Radcliffe (1991, 1994)). Nevertheless, there is overwhelming experimental evidence for the usefulness of this approach.

In this section we illustrate the case of problem-specific operators on three examples: the transportation problem, nonlinear optimization with linear constraints, and the traveling salesman problem. These three examples illustrate very well the mechanisms for incorporating problem-specific knowledge into specialized operators; in all examples operators transform feasible solutions into feasible offspring.

This is a very popular approach; most applications developed to date include some specialized operators which 'understand' the problem domain and preserve feasibility of solutions; many articles in this volume describe evolutionary systems with such operators.

C5.5.2 The transportation problem

An evolutionary system Genetic-2n for the transportation problem is discussed fully in Section G9.8. Here we concentrate on operators which have been developed in connection with Genetic-2n. G9.8

Three 'genetic' operators were defined: two mutations and one crossover. All these operators transform a matrix (representing a feasible transportation plan) into a new matrix (another feasible transportation plan). Note that a feasible matrix (i.e. a feasible transportation plan) should satisfy all marginal sums (i.e. totals for all rows and columns should be equal to given numbers, which represent supplies and demands at various sites). For example, let us assume that a transportation problem is defined with four sources and five destinations, where the supplies (vector sour) and demands (vector dest) are as follows:

$$\text{sour}[1] = 8.0 \quad \text{sour}[2] = 4.0 \quad \text{sour}[3] = 12.0 \quad \text{sour}[4] = 6.0$$

and

$$\text{dest}[1] = 3.0 \quad \text{dest}[2] = 5.0 \quad \text{dest}[3] = 10.0 \quad \text{dest}[4] = 7.0 \quad \text{dest}[5] = 5.0.$$

Then, the following matrix represents a feasible solution to the above transportation problem:

0.0	0.0	**5.0**	0.0	**3.0**
0.0	4.0	0.0	0.0	0.0
0.0	0.0	**5.0**	7.0	**0.0**
3.0	1.0	0.0	0.0	2.0

Note that the sum of all entries in the ith row is equal to sour[i] ($i = 1, 2, 3, 4$) and the total of all entries in the jth column equals to dest[j] ($j = 1, 2, 3, 4, 5$).

The first mutation selects some (random) number of rows and columns from a parent matrix; assume that the first and the third rows were selected together with the first, third, and fifth columns. The entries which are placed on the intersection of selected rows and columns (typed in boldface in the original matrix) form the following submatrix:

0.0	5.0	3.0
0.0	5.0	0.0

In this submatrix all marginal sums are calculated, and all values are reinitialized. The initialization procedure introduces as many zero entries into the matrix as possible, thus searching the surface of the feasible convex search space. All marginal totals are left unchanged. For example, the following submatrix may result after the reinitialization process is completed:

0.0	8.0	0.0
0.0	2.0	3.0

Consequently, the offspring matrix (a new feasible transportation plan) is

0.0	0.0	**8.0**	0.0	**0.0**
0.0	4.0	0.0	0.0	0.0
0.0	0.0	**2.0**	7.0	**3.0**
3.0	1.0	0.0	0.0	2.0

The only difference between the two mutation operators is that the second one avoids introducing zero while reinitializing submatrices, thus moving a feasible solution towards the center of the feasible search space.

The third operator, arithmetical crossover, for any two feasible parents (matrices U and V) produces two children X and Y, where $X = c_1U + c_2V$ and $Y = c_1V + c_2U$ (where $c_1, c_2 \geq 0$ and $c_1 + c_2 = 1$). As the constraint set is convex this operation ensures that both children are feasible if both parents are.

It is clear that all operators of Genetic-2n maintain feasibility of potential solutions: arithmetical crossover produces a point between two feasible points of the convex search space and both mutations were restricted to submatrices only to ensure no change in marginal sums.

For more discussion and the experimental results the reader is referred to Section G9.8. G9.8

C5.5.3 Nonlinear optimization with linear constraints

Let us consider the following optimization problem: optimize a function $f(x_1, x_2, \ldots, x_n)$ subject to the following sets of linear constraints:

(i) *Domain constraints.* $l_i \leq x_i \leq u_i$ for $i = 1, 2, \ldots, n$. We write $\boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u}$, where $\boldsymbol{l} = (l_1, \ldots, l_n)$, $\boldsymbol{u} = (u_1, \ldots, u_n)$, $\boldsymbol{x} = (x_1, \ldots, x_n)$.

(ii) *Equalities.* $A\boldsymbol{x} = \boldsymbol{b}$, where $\boldsymbol{x} = (x_1, \ldots, x_n)$, $A = (a_{ij})$, $\boldsymbol{b} = (b_1, \ldots, b_p)$, $1 \leq i \leq p$, and $1 \leq j \leq n$ (p is the number of equations).

(iii) *Inequalities.* $C\boldsymbol{x} \leq \boldsymbol{d}$, where $\boldsymbol{x} = (x_1, \ldots, x_n)$, $C = (c_{ij})$, $\boldsymbol{d} = (d_1, \ldots, d_m)$, $1 \leq i \leq m$, and $1 \leq j \leq n$ (m is the number of inequalities).

Due to the linearity of the constraints, the solution space is always a convex space $\mathcal{D}$. Convexity of $\mathcal{D}$ implies that:

- for any two points s_1 and s_2 in the solution space $\mathcal{D}$, the linear combination $as_1 + (1 - a)s_2$, where $a \in [0, 1]$, is a point in $\mathcal{S}$.
- for every point $s_0 \in \mathcal{S}$ and any line p such that $s_0 \in p$, p intersects the boundaries of $\mathcal{S}$ at precisely two points, say $l_p^{s_0}$ and $u_p^{s_0}$.

Consequently, the value of the ith component of a feasible solution $\boldsymbol{x} = (x_1, \ldots, x_n)$ is always in some (dynamic) range $(\text{left}(i), \text{right}(i))$; the bounds $\text{left}(i)$ and $\text{right}(i)$ depend on the other vector values $x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n$, and the set of constraints.

Several specialized operators were developed on the basis of the above properties; we discuss some of them in turn.

Uniform mutation. This operator requires a single parent $\boldsymbol{x}$ and produces a single offspring $\boldsymbol{x}'$. The operator selects a random component $k \in \{1, \ldots, n\}$ of the vector $\boldsymbol{x} = (x_1, \ldots, x_k, \ldots, x_n)$ and produces $\boldsymbol{x}' = (x_1, \ldots, x'_k, \ldots, x_n)$, where x'_k is a random value (uniform probability distribution) from the range $(\text{left}(k), \text{right}(k))$.

Boundary mutation. This operator requires also a single parent $\boldsymbol{x}$ and produces a single offspring $\boldsymbol{x}'$. The operator is a variation of the uniform mutation with x'_k being either $\text{left}(k)$ or $\text{right}(k)$, each with equal probability.

Nonuniform mutation. This is the (unary) operator responsible for the fine-tuning capabilities of the system. It is defined as follows. For a parent $\boldsymbol{x}$, if the element x_k is selected for this mutation, the result is $\boldsymbol{x}' = (x_1, \ldots, x'_k, \ldots, x_n)$, where

$$x'_k = \begin{cases} x_k + \triangle(t, \text{right}(k) - x_k) & \text{if a random binary digit is 0} \\ x_k - \triangle(t, x_k - \text{left}(k)) & \text{if a random binary digit is 1.} \end{cases}$$

The function $\triangle(t, y)$ returns a value in the range $[0, y]$ such that the probability of $\triangle(t, y)$ being close to zero increases as t increases (t is the generation number). This property causes this operator to search the space uniformly initially (when t is small), and very locally at later stages. We have used the following function:

$$\triangle(t, y) = yr(1 - t/T)^b$$

where r is a random number in the range [0..1], T is the maximal generation number, and b is a system parameter determining the degree of nonuniformity.

Arithmetical crossover. This binary operator is defined as a linear combination of two vectors: if $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ are to be crossed, the resulting offspring are $\boldsymbol{x}'_1 = a\boldsymbol{x}_1 + (1 - a)\boldsymbol{x}_2$ and $\boldsymbol{x}'_2 = a\boldsymbol{x}_2 + (1 - a)\boldsymbol{x}_1$. This operator uses a random value $a \in [0..1]$, as it always guarantees closedness $(\boldsymbol{x}'_1, \boldsymbol{x}'_2 \in \mathcal{D})$.

All of the above operators preserve feasibility, transforming a feasible parent(s) into feasible offspring. For more information on these and additional operators, as well as experimental results of the implemented system, see Section G9.1. G9.1

C5.5.4 Traveling salesman problem

Whitley *et al* (1989) developed the *edge recombination* crossover (ER) for the *traveling salesman problem*. G9.5 The ER operator explores the information on edges in a tour, for example for the tour

(3 1 2 8 7 4 6 9 5)

the edges are (3 1), (1 2), (2 8), (8 7), (7 4), (4 6), (6 9), (9 5), and (5 3). After all, edges—not cities—carry values (distances) in the TSP. The objective function to be minimized is the total of edges which constitute a legal tour. The position of a city in a tour is not important: tours are circular. Also, the direction of an edge is not important: both edges (3 1) and (1 3) signal only that cities 1 and 3 are directly connected.

The general idea behind the ER crossover is that an offspring should be built exclusively from the edges present in both parents. This is done with help of the edge list created from both parent tours. The edge list provides, for each city c, all other cities connected to city c in at least one of the parents.

Obviously, for each city c there are at least two and at most four cities on the list. For example, for the two parents

$$p_1 = (1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9)$$

and

$$p_2 = (4\ 1\ 2\ 8\ 7\ 6\ 9\ 3\ 5)$$

the edge list is

city 1 : edges to other cities: 9 2 4
city 2 : edges to other cities: 1 3 8
city 3 : edges to other cities: 2 4 9 5
city 4 : edges to other cities: 3 5 1
city 5 : edges to other cities: 4 6 3
city 6 : edges to other cities: 5 7 9
city 7 : edges to other cities: 6 8
city 8 : edges to other cities: 7 9 2
city 9 : edges to other cities: 8 1 6 3.

The construction of the offspring starts with a selection of an initial city from one of the parents. Whitley *et al* (1989) selected one of the initial cities (e.g. 1 or 4 in the example above). The city with the smallest number of edges in the edge list is selected. If these numbers are equal, a random choice is made. Such selection increases the chance that we complete a tour with all edges selected from the parents. With a random selection, the chance of having edge failure, that is, being left with a city without a continuing edge, would be much higher. Assume we have selected city 1. This city is directly connected with three other cities: 9, 2, and 4. The next city is selected from these three. In our example, cities 4 and 2 have three edges, and city 9 has four. A random choice is made between cities 4 and 2; assume city 4 was selected. Again, the candidates for the next city in the constructed tour are 3 and 5, since they are directly connected to the last city, 4. Again, city 5 is selected, since it has only three edges as opposed to the four edges of city 3. So far, the offspring has the following shape:

$$(1\ 4\ 5\ x\ x\ x\ x\ x\ x).$$

Continuing this procedure we finish with the offspring

$$(1\ 4\ 5\ 6\ 7\ 8\ 2\ 3\ 9)$$

which is composed entirely of edges taken from the two parents.

The ER crossover was further enhanced (Starkweather *et al* 1991). The idea was that the 'common subsequences' were not preserved in the ER crossover. For example, if the edge list contains the row with three edges

city 4 : edges to other cities: 3 5 1

then one of these edges repeats itself. Referring to the previous example, it is the edge (4 5). This edge is present in both parents. However, it is listed as other edges, for example (4 3) and (4 1), which are present in one parent only. The proposed solution modifies the edge list by storing 'flagged' cities:

city 4 : edges to other cities: 3 -5 1.

The notation '-' means simply that the flagged city 5 should be listed twice. In the previous example of two parents

$$p_1 = (1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9)$$

and

$$p_2 = (4\ 1\ 2\ 8\ 7\ 6\ 9\ 3\ 5)$$

the (enhanced) edge list is:

city 4 : edges to other cities: 9 -2 4
city 4 : edges to other cities: -1 3 8
city 4 : edges to other cities: 2 4 9 5
city 4 : edges to other cities: 3 -5 1
city 4 : edges to other cities: -4 6 3
city 4 : edges to other cities: 5 -7 9
city 4 : edges to other cities: -6 -8
city 4 : edges to other cities: -7 9 2
city 4 : edges to other cities: 8 1 6 3.

The algorithm for constructing a new offspring gives priority to flagged entries: this is important only in the cases where three edges are listed—in the two other cases either there are no flagged cities, or both cities are flagged. This enhancement (plus a modification for making better choices when random edge selection is necessary) further improved the performance of the system.

We illustrate this enhancement using the previous example. Assume we have selected city 1 as an initial city for an offspring. As before, this city is directly connected with three other cities: 9, 2, and 4. In this case, however, city 2 is flagged, so it is selected as the next city of the tour. Again, the candidates for the next city in the constructed tour are 3 and 8, since they are directly connected to the last city, 2; the flagged city 1 is already present in the partial tour and is not considered. Again, city 8 is selected, since it has only three edges as opposed to the four edges of city 3. So far, the offspring has the following shape:

(1 2 8 x x x x x x).

Continuing this procedure we finish with the offspring

(1 2 8 7 6 5 4 3 9)

which is composed entirely of edges taken from the two parents.

References

Davis L (ed) 1987 *Genetic Algorithms and Simulated Annealing* (Los Altos, CA: Morgan Kaufmann)

——1991 *Handbook of Genetic Algorithms* (New York: Van Nostrand Reinhold)

Radcliffe N J 1991 Forma analysis and random respectful recombination *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, July 1991)* ed R Belew and L Booker (San Mateo, CA: Morgan Kaufmann) pp 222–9

Radcliffe N J 1994 The algebra of genetic algorithms *Ann. Maths Artificial Intell.* **10** 339–84

Starkweather T, McDaniel S, Mathias K, Whitley C and Whitley D 1991 A comparison of genetic sequencing operators *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, July 1991)* ed R Belew and L Booker (San Mateo, CA: Morgan Kaufmann) pp 69–76

Surry P D, Radcliffe N J and Boyd I D 1995 A multi-objective approach to constrained optimization of gas supply networks *AISB-95 Workshop on Evolutionary Computing (Sheffield, 1995)* ed T Fogarty (Berlin: Springer) pp 166-80

Whitley D, Starkweather T and Fuquay D'A 1989 Scheduling problems and traveling salesman: the genetic edge recombination operator *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, June 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 133–40

C5.6 Other constraint-handling methods

Zbigniew Michalewicz

Abstract

This section discusses several additional constraint-handling methods which have been proposed in connection with evolutionary techniques.

C5.6.1 Introduction

Several additional constraint-handling heuristics have emerged during the last few years. Often these methods are difficult to classify: either they are based on some new ideas or they combine a few elements present in other methods. In this section several such techniques are discussed.

C5.6.2 Multiobjective optimization methods

One of the techniques includes utilization of multiobjective optimization methods, where the objective function f and, for m constraints

$$g_j(\boldsymbol{x}) \leq 0 \qquad \text{for } j = 1, \ldots, q$$

and

$$h_j(\boldsymbol{x}) = 0 \qquad \text{for } j = q+1, \ldots, m$$

their constraint violation measures f_j

$$f_j(\boldsymbol{x}) = \begin{cases} \max\{0, g_j(\boldsymbol{x})\} & \text{if } 1 \leq j \leq q \\ |h_j(\boldsymbol{x})| & \text{if } q+1 \leq j \leq m \end{cases}$$

constitute an $(m+1)$-dimensional vector:

$$\text{eval}(\boldsymbol{x}) = (f, f_1, \ldots, f_m).$$

Using some multiobjective optimization method, we can attempt to minimize its components: an ideal solution x would have $f_j(x) = 0$ for $1 \leq i \leq m$ and $f(x) \leq f(y)$ for all feasible y (minimization problems).

A successful implementation of a similar approach was presented recently by Surry *et al* (1995). All individuals in the population are measured with respect to constraint satisfaction—each individual $\boldsymbol{x}$ is assigned a rank $r(\boldsymbol{x})$ according to its Pareto ranking; the rank can be assigned either by peeling off successive nondominating layers or by calculating the number of solutions which dominate it (see Section C4.5 for more details on multiobjective optimization). Then, the evaluation measure of each individual is given as a two-dimensional vector:

C4.5

$$\text{eval}(\boldsymbol{x}) = \langle f(\boldsymbol{x}), r(\boldsymbol{x}) \rangle.$$

At this stage, a modified Schaffer (1984) VEGA system (for vector evaluated genetic algorithm) was used. The main idea behind the VEGA system was a division of the population into (equal-sized)

subpopulations; each subpopulation was 'responsible' for a single objective. The selection procedure was performed independently for each objective, but crossover was performed across subpopulation boundaries. Additional heuristics were developed (e.g. wealth redistribution scheme, crossbreeding plan) and studied to decrease a tendency of the system to converge towards individuals which were not the best with respect to any objective. However, instead of proportional selection, a binary tournament selection was used, where the tournament criterion cost value f is selected with probability p and constraint ranking r with probability $1 - p$. The value of the parameter p is adapted during the run of the algorithm; it is increased or decreased on the basis of the ratio of feasible individuals in the recent generations (i.e. if the ratio is too low, the parameter p is decreased. Clearly, if p approaches zero, the system favors constraint rank). The outline of the system implemented for a particular problem of the optimization of gas supply networks (Surry *et al* 1995) is as follows:

- calculate constraint violation for all solutions
- rank individuals based on constraint violation (Pareto ranking)
- evaluate the cost of solutions (in terms of function f)
- select proportion p of parents based on cost, and the other based on ranking
- perform genetic operators
- adjust p on the basis of the ratio of feasible individuals in the recent generations.

C5.6.3 Coevolutionary model approach

Another approach was reported by Paredis (1994). The method (described in the context of constraint satisfaction problems) is based on a coevolutionary model, where a population of potential solutions coevolves with a population of constraints: fitter solutions satisfy more constraints, whereas fitter constraints are violated by more solutions (i.e. the harder it is to satisfy a constraint, the fitter the constraint is, thus participating more actively in evaluating individuals from the solution space). This means that individuals from the population of solutions are considered from the whole search space $\mathcal{S}$, and that there is no distinction between feasible and infeasible individuals (i.e. there is only one evaluation function eval without any split into eval_f for feasible or eval_u for infeasible individuals). The value of eval is determined on the basis of constraint violation measures f_j; however, fitter constraints (e.g. active constraints) would contribute more frequently to the value of eval.

Yet another heuristic is based on the idea of handling constraints in a particular order; Schoenauer and Xanthakis (1993) called this method a 'behavioral memory' approach. The initial steps of the method are devoted to sampling the feasible region; only in the final step is the objective function f optimized.

- Start with a random population of individuals (i.e. these individuals are feasible or infeasible).
- Set $j = 1$ (j is the constraint counter).
- Evolve this population to minimize the violation of the jth constraint, until a given percentage of the population (the so-called flip threshold ϕ) is feasible for this constraint. In this case

$$\text{eval}(\boldsymbol{x}) = g_1(\boldsymbol{x}).$$

- Set $j = j + 1$.
- The current population is the starting point for the next phase of the evolution, minimizing the violation of the jth constraint:

$$\text{eval}(\boldsymbol{x}) = g_j(\boldsymbol{x}).$$

  (To simplify notation, we do not distinguish between inequality constraints g_j and equations h_j; all m constraints are denoted by g_j.) During this phase, points that do not satisfy at least one of the first, second, ..., $(j-1)$th constraints are eliminated from the population. The halting criterion is again the satisfaction of the jth constraint by the flip threshold percentage ϕ of the population.
- If $j < m$, repeat the last two steps, otherwise ($j = m$) optimize the objective function f rejecting infeasible individuals.

The method has a few merits. One of them is that in the final step of the algorithm the objective function f is optimized (as opposed to its modified form). However, for larger feasible spaces the method just provides additional computational overhead, and for very small feasible search spaces it is essential to maintain diversity in the population.

C5.6.4 Cultural algorithms

It is also possible to incorporate the knowledge of the constraints of the problem into the belief space of cultural algorithms (Reynolds 1994); such algorithms provide a possibility of conducting an efficient search of the feasible search space (Reynolds *et al* 1995). The research on cultural algorithms (Reynolds 1994) was triggered by observations that culture might be another kind of inheritance system. However it is not clear what the appropriate structures and units to represent the adaptation and transmission of cultural information are. Neither is it clear how to describe the interaction between natural evolution and culture. Reynolds developed a few models to investigate the properties of cultural algorithms; in these models, the belief space is used to constrain the combination of traits that individuals can assume. Changes in the belief space represent macroevolutionary change and changes in the population of individuals represent microevolutionary change. Both changes are moderated by the communication link.

The general intuition behind belief spaces is to preserve those beliefs associated with 'acceptable' behavior at the trait level (and, consequently, to prune away unacceptable beliefs). The acceptable beliefs serve as constraints that direct the population of traits. It seems that the cultural algorithms may serve as a very interesting tool for numerical optimization problems, where constraints influence the search in a direct way (consequently, the search may be more efficient in constrained spaces than in unconstrained ones!).

C5.6.5 Segregated genetic algorithm

Le Riche *et al* (1995) proposed a method which combines the ideas of penalizing infeasible solutions with coevolutionary concepts. The 'classical' methods based on penalty functions either (i) maintain static penalty coefficients (see e.g. Homaifar *et al* 1994), (ii) use dynamic penalties (Smith and Tate 1993), (iii) use penalties which are functions of the evolution time (Michalewicz and Attia 1994, Joines and Houck 1994), or (iv) adapt penalty coefficients on the basis of the number of feasible and infeasible individuals in recent generations (Bean and Hadj-Alouane 1992). The method of Le Riche proposes a so-called segregated genetic algorithm which uses a double-penalty strategy. The population is split into two coevolving subpopulations, where the fitness of each subpopulation is evaluated using either one of the two penalty parameters. The two subpopulations converge along two complementary trajectories, which may help to locate the optimal region faster. Also, such a system may make the algorithm less sensitive to the choice of penalty parameters.

The outline of the method is as follows:

- create two sets of penalty coefficients, p_j and r_j $(j = 1, \ldots, m)$, where $p_i \ll r_i$
- start with two random populations of individuals (i.e. these individuals are feasible or infeasible); the size of each population is the same pop_size
- evaluate these populations; the first population is evaluated as

$$\text{eval}(\boldsymbol{x}) = f(\boldsymbol{x}) + \sum_{j=1}^{m} p_j f_j^2(\boldsymbol{x})$$

  and the other population as

$$\text{eval}(\boldsymbol{x}) = f(\boldsymbol{x}) + \sum_{j=1}^{m} r_j f_j^2(\boldsymbol{x})$$

- create two separate ranked lists
- merge the two lists into one ranked population of the size pop_size
- apply selection and operators to the new population; create the new population of pop_size offspring
- evaluate the new population twice (with respect to p_j and r_j values, respectively)
- from the old and new populations create two populations of the size pop_size each; each population is ranked accordingly to its evaluation with respect to p_j and r_j values, respectively
- repeat the last four steps.

The major merit of this method is that it permits the balancing of the influence of two sets of penalty parameters. Note that if $p_j = r_j$ (for $1 \leq j \leq m$), the algorithm is similar to the (pop_size, pop_size) *evolution strategy*. The reported results of applying the above segregated genetic algorithm to the laminate design problem were very good (Le Riche *et al* 1995). B1.3

C5.6.6 Genocop III

Michalewicz and Nazhiyath (1995) mixed the idea of repair algorithms with concepts of coevolution. The *Genocop III* system maintains two separate populations: the first population consists of (not necessarily feasible) search points and the second population consists of fully feasible reference points. Reference points, being feasible, are evaluated directly by the objective function. Search points are 'repaired' for evaluation. The repair process (described in detail in Section G9.9) samples the segment between the search and reference points; the first feasible point is accepted as a repaired version of the search point. Genocop III avoids some disadvantages of other systems. It uses the objective function for evaluation of fully feasible individuals only, so the evaluation function is not distorted as in methods based on penalty functions. It introduces only few additional parameters and it always returns a feasible solution. However, it requires an efficient repair process, which might be too costly for many engineering problems.

G9.9.3

G9.9

A comparison of some of the above methods is presented by Michalewicz (1995).

References

Bean J C and Hadj-Alouane A B 1992 *A Dual Genetic Algorithm for Bounded Integer Programs* Department of Industrial and Operations Engineering, University of Michigan, TR 92-53

Hadj-Alouane A B and Bean J C 1992 *A Genetic Algorithm for the Multiple-Choice Integer Program* Department of Industrial and Operations Engineering, University of Michigan, TR 92-50

Homaifar A, Lai S H-Y and Qi X 1994 Constrained optimization via genetic algorithms *Simulation* **62** 242–54

Juliff K 1993 A multi-chromosome genetic algorithm for pallet loading *Proc. 5th Int. Conf. on Genetic Algorithms* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 467–73

Le Riche R G, Knopf-Lenoir C and Haftka R T 1995 A segregated genetic algorithm for constrained structural optimization *Proc. 6th Int. Conf. on Genetic Algorithms* ed L Eshelman (San Mateo, CA: Morgan Kaufmann) pp 558–65

Michalewicz Z 1995 Genetic algorithms, numerical optimization and constraints *Proc. 6th Int. Conf. on Genetic Algorithms* ed L Eshelman (San Mateo, CA: Morgan Kaufmann) pp 151–8

Michalewicz Z and Attia N 1994 Evolutionary optimization of constrained problems *Proc. 3rd Ann. Conf. on Evolutionary Programming* ed A V Sebald and L J Fogel (Singapore: World Scientific) pp 98–108

Michalewicz Z and Nazhiyath G 1995 Genocop III: a co-evolutionary algorithm for numerical optimization problems with nonlinear constraints *Proc. 2nd IEEE Int. Conf. on Evolutionary Computation (Perth, 1995)* pp 647–51

Paredis J 1994 Co-evolutionary constraint satisfaction *Proc. 3rd Conf. on Parallel Problem Solving from Nature* (New York: Springer) pp 46–55

Reynolds R G 1994 An introduction to cultural algorithms *Proc. 3rd Ann. Conf. on Evolutionary Programming* ed A V Sebald and L J Fogel (Singapore: World Scientific) pp 131–9

Reynolds R G, Michalewicz Z and Cavaretta M 1995 Using cultural algorithms for constraint handling in Genocop *Proc. 4th Ann. Conf. on Evolutionary Programming (San Diego, CA, 1995)* ed J R McDonnell, R G Reynolds and D B Fogel pp 289–305

Schaffer J D 1984 *Some Experiments in Machine Learning Using Vector Evaluated Genetic Algorithms* PhD Dissertation, Vanderbilt University

Schoenauer M and Xanthakis S 1993 Constrained GA optimization *Proc. 5th Int. Conf. on Genetic Algorithms* ed S Forrest (Los Altos, CA: Morgan Kaufmann) pp 573–80

Smith A E and Tate D M 1993 Genetic optimization using a penalty function *Proc. 5th Int. Conf. on Genetic Algorithms* ed S Forrest (Los Altos, CA: Morgan Kaufmann) pp 499–503

Surry P D, Radcliffe N J and Boyd I D 1995 A multi-objective approach to constrained optimization of gas supply networks *AISB-95 Workshop on Evolutionary Computing (Sheffield, 1995)*

C5.7 Constraint-satisfaction problems

A E Eiben and Zs Ruttkay

Abstract

In this section we discuss solving constraint-satisfaction problems with evolutionary algorithms. We set up a formal framework by defining the notions of free optimization problem, constrained optimization problem, and constraint-satisfaction problem. We note that constraint-satisfaction problems (CSPs) lack an optimization component that is necessary for evolutionary problem solvers. Therefore we discuss different ways of solving CSPs in two stages: applying a suitable problem transformation and solving the transformed problem.

C5.7.1 Introduction

Applying evolutionary algorithms (EAs) for solving constraint-satisfaction problems (CSP) is interesting from two points of view. On the one hand, since a general CSP is known to be NP-complete (Mackworth 1977), one cannot expect that an effective classical deterministic search algorithm can be forged to solve CSPs. There has been a continuous effort to construct effective algorithms for specific CSPs, and to characterize the difficulty of problem classes. The proposed algorithms apply different search strategies, often guided by heuristics based on some evaluation of the uninstantiated variables and of the possible values. The search can be preceded by preprocessing of the domains or the constraints. For an overview of specific search strategies and heuristics see the work of Meseguer (1989), Nudel (1983), and Tsang (1993). What makes deterministic heuristic search methods strong in certain cases is just what makes them weak in others: they restrict the scope of the search, based on (explicit or implicit) heuristics. If the heuristics turn out to be misleading, it is often very tiresome to enlarge or shift the scope of the search using a series of backtrackings. This problem could be treated by diversifying the search by maintaining several different candidate solutions in parallel and counterbalancing the greediness of the heuristics by incorporating random elements into the construction mechanism of new candidates. These two principles are essential for EAs. Hence, the idea of applying EAs to solve CSPs is a natural response for the limitations of the classical CSP solving methods.

On the other hand, traditional EAs are mainly used for unconstrained optimization. Problems where constraints play an essential role have the common reputation of being EA hard. This is due to the fact that standard genetic operators (mutation and crossover) are ‘blind’ to constraints. In other words, there is no guarantee that children of feasible parents are also feasible, nor that children of infeasible parents are ‘less infeasible’ than the parents. Thus, handling constrained problems with EAs is a big challenge for the field (cf Michalewicz and Michalewicz 1995).

The main goals of this section are to present definitions that yield a clear conceptual framework and terminology for constrained problems and to discuss different ways to apply EAs for solving CSPs within the above framework.

C5.7.2 Free optimization, constrained optimization, and constraint satisfaction

Let us first set up a general conceptual framework for constrained problems. Without such a framework terminology and views can be ambiguous. For instance, the *traveling salesperson problem* (TSP) is a constrained problem if we consider that each variable can have each city label as a value but a solution can contain each label only once. This latter restriction is a constraint on the search space $S = D_1 \times \ldots \times D_n$, G9.5

where each D_i ($i \in \{1, \ldots, n\}$) is the set of all city labels. Nevertheless, the TSP is an unconstrained problem if we define the search space as the set of all permutations of the city labels.

Definition C5.7.1. We will call a Cartesian product of sets $S = D_1 \times \ldots \times D_n$ a *free search space*.

Note that this definition puts no requirements on the domains; they can be discrete or continuous, connected or not. The rationale behind this definition is that testing the membership relation of a free search space can be performed independently on each coordinate and taking the conjunction of the results. This implies an interesting property from an EA point of view: if two chromosomes from a free search space are crossed over, their offspring will be in the same space as well. Thus, (genetic) search in a space of the form $S = D_1 \times \ldots \times D_n$ is free in this sense; this motivates the name.

Definition C5.7.2. A *free optimization problem* (FOP) is a pair $\langle S, f \rangle$, where S is a free search space and f is a (real-valued) objective function on S, which has to be optimized (minimized or maximized). A *solution of a free optimization problem* is an $s \in S$ with an optimal f-value.

Definition C5.7.3. A *constraint-satisfaction problem* (CSP) is a pair $\langle S, \phi \rangle$, where S is a free search space and ϕ is a formula (a Boolean function on S). A *solution of a constraint-satisfaction problem* is an $s \in S$ with $\phi(s) = true$.

Usually a CSP is stated as a problem of finding an instantiation of variables $v_1, \ldots, v_n$ within the finite domains $D_1, \ldots, D_n$ such that constraints (relations) $c_1, \ldots, c_m$ prescribed for (some of) the variables hold. The formula ϕ is then the conjunction of the given constraints. One may be interested in one, some, or all solutions, or only in the existence of a solution. We restrict our discussion to finding one solution. It is also worth noting that our definitions allow CSPs with continuous domains. Such a case is almost never considered in the CSP literature: by the finiteness assumption on the domains $D_1, \ldots, D_n$ the usual CSPs are discrete.

Definition C5.7.4. A *constrained optimization problem* (COP) is a triple $\langle S, f, \phi \rangle$, where S is a free search space, f is a (real-valued) objective function on S and ϕ is a formula (a Boolean function on S). A *solution of a constrained optimization problem* is an $s \in S$ with $\phi(s) = true$ and an optimal f-value.

The above three problem types can be represented in the same scheme as $\langle S, f, \bullet \rangle$, $\langle S, \bullet, \phi \rangle$, and $\langle S, f, \phi \rangle$ respectively, where $\bullet$ means the absence of the given component.

Definition C5.7.5. For CSPs, as well as for COPs, we call the ϕ the *feasibility condition*, and the set $\{s \in S | \phi(s) = true\}$ will be called the *feasible search space*.

With this terminology, solving a CSP means finding one single feasible element; solving a COP means finding a feasible and optimal element. These definitions eliminate the arbitrariness of viewing a problem as a constrained problem. For example, in the most natural formalization of the TSP candidate solutions are permutations of the cities $\{x_1, \ldots, x_n\}$. The TSP is then a COP $\langle S, f, \phi \rangle$, where $S = \{x_1, \ldots, x_n\}^n$, $\phi(s) = true \Leftrightarrow \forall i, j \in \{1, \ldots, n\}\ s_i \neq s_j$ and $f(s) = \sum_{i=1}^{n} \mathrm{dist}(s_i, s_{i+1})$, where $s_{n+1} := s_1$.

C5.7.3 Transforming constraint-satisfaction problems to evolutionary-algorithm-suited problems

Note that the presence of an objective function (fitness function) to be optimized is essential for EAs. A CSP $\langle S, \bullet, \phi \rangle$ is lacking this component; in this sense it is not EA suited. Therefore it needs to be transformed to an EA-suited problem, an FOP $\langle S, f, \bullet \rangle$ or a COP $\langle S, f, \psi \rangle$, before an EA can be applied to it.

Definition C5.7.6. Let problems A and B be either of $\langle S, f, \bullet \rangle$, $\langle S, \bullet, \phi \rangle$, $\langle S, f, \phi \rangle$. A and B are *equivalent* if

$$\forall s \in S:\ s \text{ is a solution of } A \Longleftrightarrow s \text{ is a solution of } B.$$

We say that A *subsumes* B if

$$\forall s \in S:\ s \text{ is a solution of } A \Longrightarrow s \text{ is a solution of } B.$$

Thus, solving a CSP by an EA means that we transform it to an FOP/COP that subsumes it and solve this FOP/COP. In the sequel we discuss how to transform CSPs to FOPs and COPs. Solving the resulting problems is another issue. FOPs allow free search; in this sense they are simple for an EA; COPs, however,

are difficult to solve by EAs. Notice that the term 'constraint handling' has two meanings in this context. Its first meaning is 'how to transform the constraints of a given CSP': whether, and how, to incorporate them in an FOP or a COP. The second meaning emerges when a CSP $\longrightarrow$ COP transformation is chosen: 'how to maintain the constraints when solving a COP by an EA'. It is this second meaning that is mostly intended in the EA literature.

C5.7.3.1 Transforming a constraint-satisfaction problem to a free optimization problem

Let $\langle S, \bullet, \phi \rangle$ be a CSP and let us assume that ϕ is given by a conjunction of some constraints (relations) $c_1, \ldots, c_m$ that have to hold for the variables. An equivalent FOP can be created by defining an objective function f which has an optimal value if and only if all constraints $c_1, \ldots c_m$ are satisfied. Applying an EA for this FOP means that all constraints are handled *indirectly*, that is the EA operates freely on S (see the remark after definition C5.7.1) and reaching the optimum means satisfying all constraints. Using such a CSP $\longrightarrow$ FOP transformation implies that 'constraint handling' is restricted to its first meaning.

Incorporating all constraints in f can be done by applying penalties on constraint violation. The most straightforward possibility for a penalty function f based on the constraints is to consider the number of violated constraints. This measure, however, does not distinguish between difficult and easy constraints. These aspects can be reflected by assigning weights to the constraints and defining f as

$$f(s) = \sum_{i=1}^{m} w_i \times \chi(s, c_i) \tag{C5.7.1}$$

where w_i is the penalty (or weight) assigned to constraint c_i and

$$\chi(s, c_i) = \begin{cases} 1 & \text{if } s \text{ violates } c_i \\ 0 & \text{otherwise.} \end{cases}$$

Satisfying a constraint with a high penalty gives a relatively high reward to the EA, hence it will be 'more interested' in satisfying such constraints. Thus, the definition of an appropriate *penalty function* is of crucial importance. For determining the constraint weights one can use the measures common in classical CSP solving methods (e.g. constraint tightness) to evaluate the difficulty of constraints. A more sophisticated notion of penalties can be based on the degree of violation for each constraint. In this approach $\chi(s, c_i)$ is not simply 1 or 0, but a measure of how severe the violation of c_i is.

C5.2

Another type of penalty function is obtained if we concentrate on the variables, instead of the constraints. The function f is then based on the evaluation of the incorrect values, that is variables where the value violates at least one constraint. Let C^i ($i \in \{1, \ldots, n\}$) be the set of constraints that involves variable i. Then

$$f(s) = \sum_{i=1}^{n} w_i \times \chi(s, C^i) \tag{C5.7.2}$$

where w_i is the penalty (or weight) assigned to variable i and

$$\chi(s, C^i) = \begin{cases} 1 & \text{if } s \text{ violates at least one } c \in C^i \\ 0 & \text{otherwise.} \end{cases}$$

Obviously, this approach can also be refined by measuring how serious the violation of constraints is. A particular implementation of this idea is to define $\chi(s, C^i)$ as $\sum_{c_j \in C^i} \chi(s, c_j)$.

Example C5.7.1. Consider the graph three-coloring problem, where the nodes of a given graph $G = (N, E)$, $E \subseteq N \times N$, have to be colored by three colors in such a way that no neighboring nodes, i.e. nodes connected by an edge, have the same color. Formally we can represent this problem as a CSP with $n = |N|$ variables, each with the same domain $D = \{1, 2, 3\}$. Furthermore, we need $m = |E|$ constraints, each belonging to one edge, with $c_e(s) = true$ iff $e = (k, l)$ and $s_k \neq s_l$, i.e. the two nodes on the corresponding edge have a different color. Then the corresponding CSP is $\langle S, \phi \rangle$, where $S = D^n$ and $\phi(s) = \bigwedge_{e \in E} c_e$. Using a constraint-oriented penalty function (equation (C5.7.1)) with $w_e \equiv 1$ we would count the incorrect edges that connect two nodes with the same color. The variable-oriented penalty function (equation (C5.7.2)) with $w_i \equiv 1$ amounts to counting the incorrect nodes that have a neighbor with the same color.

A great many penalty functions used in practice are (a variant of) one of the above two options. There are, however, other possibilities. For instance, instead of viewing the objective function f as a penalty for constraint violation, it can be perceived as the distance to a solution, or more generally, the cost of reaching a solution. In order to define such an f a distance measure d on the search space has to be given. Since the solutions are not known in advance, the real distance from $s \in S$ to the set of solutions can only be estimated. One such an estimation is based on the *projection of a constraint c*. If c operates on the variables $v_{i_1}, \ldots, v_{i_k}$, then this projection is defined as the set $S_c = \{\langle s_{i_1}, \ldots, s_{i_k}\rangle \in D_{i_1} \times \ldots \times D_{i_k} \mid c(s_{i_1}, \ldots, s_{i_k}) = true\}$, and the distance of $s \in S$ from S_c is $d(s, S_c) := \min\{d(\langle s_{i_1}, \ldots, s_{i_k}\rangle, z) \mid z \in S_c\}$. (We assume that d is also defined on hyperplanes of S.) Now it is a natural idea to estimate the distance of an $s \in S$ from a solution as

$$f(s) = \sum_{i=1}^{m} w_i \times d(s, S_{c_i}). \tag{C5.7.3}$$

It is clear that $s \in S$ is a solution of the CSP iff $f(s) = 0$. Function (C5.7.3) is just an example satisfying this equivalence property. In general an objective function of the third kind does not necessarily have to be based on a distance measure d. It can be any cost measure as long as $f(s) = 0$ implies membership of the set of solutions.

Example C5.7.2. Consider the graph three-coloring problem again. The projection of a constraint $c_{(k,l)}$ belonging to an edge (k, l) is $S_{c_{(k,l)}} = \{\langle 1, 2\rangle, \langle 1, 3\rangle, \langle 2, 1\rangle, \langle 2, 3\rangle, \langle 3, 1\rangle, \langle 3, 2\rangle\}$ and we can define the cost of reaching $S_{c_{(k,l)}}$ from an $s \in S$ as the number of value modifications in s needed to have $\langle s'_k, s'_l\rangle \in S_{c_{(k,l)}}$. In this simple example this will be one for every $s \in S$ and $c_{(k,l)}$, thus the formulas (C5.7.1) and (C5.7.3) will coincide, both counting the incorrect edges.

As we have already mentioned, solving an FOP is 'easy' for an EA, at least it is 'only' a matter of optimization. Whether or not the FOP approach is successful depends on the EA's ability to find an optimum. Penalizing infeasible individuals is studied extensively by Michalewicz (1995), primarily in the context of treating COPs with continuous domains. Experiments reported for example by Richardson *et al* (1989) and Michalewicz (1996, p 86–7) indicate that GAs with penalty functions are likely to fail on sparse problems, i.e. on problems where the ratio between the size of the feasible and the whole search space is small.

C5.7.3.2 Transforming a constraint-satisfaction problem to a constrained optimization problem

The limitations of using penalty functions as the only means of handling constraints force one to look for other options. The basic idea is to incorporate only some of the constraints in f (these are handled indirectly) and maintain the other ones *directly*. This means that the CSP $\langle S, \bullet, \phi\rangle$ is transformed to a COP $\langle S, f, \psi\rangle$, where the constraints not in f form ψ. In this case we presume that the EA works with individuals satisfying ψ, that is, it will operate on the space $\{x \in S | \psi(x) = true\}$ and finding an $s \in \{x \in S | \phi(x) = true\}$ (a solution for the CSP) means finding an $s \in \{x \in S | \psi(x) = true\}$ with an optimal f-value.

Definition C5.7.7. If the context requires a clear distinction between ϕ (expressing the constraints given in the original CSP) and ψ (expressing an appropriate subset of the constraints in the COP) we will call ψ the *allowability condition* and $\{s \in S | \psi(s) = true\}$ the *allowable search space*.

For a given CSP several equivalent COPs can be defined by choosing the subset of the constraints incorporated in the allowability condition and/or defining the objective function measuring the satisfaction of the remaining constraints differently. Such decisions can be based on distiguishing constraints for which finding and maintaining solutions is easy (in ψ) or difficult (in f) (cf Smith and Tate 1993). For the constraints in the allowability condition it is essential that they can be satisfied by the initialization procedure and can be maintained by the EA. The latter requires that the EA guarantees that the new candidate solutions are always allowable. This implies that the COP approach is more complex than the FOP approach. When deciding which constraints to incorporate in the allowability condition, one should already consider how it can be maintained by the EA in mind.

C5.7.3.3 Changing the search space

Up to now we have assumed that the candidate solutions of the original CSP and the individuals of the EA are of the same type, namely members of S. This is not necessary: the EA may operate on a different search space, S', assuming that a *decoder* is given which transforms a given genotype $s' \in S'$ C5.3 into a phenotype $s \in S$. Usually, this technique is used in such a way that the elements of S generated by decoding elements of S' automatically fulfill some of the original constraints. However, it is not guaranteed automatically that the decoder can create the whole S from S', hence, it can occur that not all solutions of the original CSP can be produced.

Here again, choosing an appropriate representation, i.e. S', for the EA and making the decoder are highly correlated. Designing a good decoder is clearly a problem specific task. However, order-based representation, where S' consists of permutations, is a generally advisable option. Many decoders create a search object (an $s \in S$) by a sequential procedure, following a certain order of processing. In these terms the goal of the search is to find a sequence that encodes a solution; that is, we have a sequencing problem. In the meanwhile, sequencing problems can be naturally represented by permutations as chromosomes and there are many off-the-shelf order-based operators at our disposal (Olivier *et al* 1987, Fox and McMahon 1991, Starkweather *et al* 1991). This makes order-based representation a promising option.

Example C5.7.3. For the graph three-coloring problem each permutation of nodes can be assigned a coloring by a simple greedy decoding algorithm. The decoder processes the nodes in the order they appear in the permutation π and colors node π_i with the smallest color from $\{1, 2, 3\}$ that does not violates the constraints. If none of the colors in $\{1, 2, 3\}$ is suitable a random assignment is made. Formally, we change to a COP $\langle S', f, \psi \rangle$, where $S' = \{s_1, \ldots, s_n\}^n$, $\psi(s') = true \Leftrightarrow \forall i, j \in \{1, \ldots, n\}\ s'_i \neq s'_j$ and f is an objective function taking its optimum on permutations that encode feasible colorings. Note that when coloring a node π_i some neighbors of i might not have a color yet, thus not all constraints can be evaluated. This means that the decoder considers a color suitable if it does not violates that subset of the constraints that can be evaluated at the given moment.

An interesting variation of this permutations + decoder approach is decoding permutations to *partial* solutions. In the work of Eiben *et al* (1994) and Eiben and van der Hauw (1996) the decoder leaves nodes uncolored in case of violations and f is the number of uncolored nodes. It might seem that this objective function supplies too little information, but the experiments showed that this EA consistently outperforms the one with integer representation (see example C5.7.1). This is even more interesting if we take into account that the permutation space has $n!$ elements, while the size of $S = \{1, 2, 3\}^n$ is only 3^n.

C5.7.4 Solving the transformed problem

If we have transformed a given CSP to an FOP we can simply apply the usual EA machinery to find a solution of this FOP. However, there can be many ways to enhance a simple function optimizing EA by incorporating constraint-specific features into it. Next we discuss two domain independent options, concerning the search operators, respectively the fitness function. One option is thus to use specific search operators based on heuristics that try to create children that are 'less infeasible' than the parents. The heuristic operators used by Eiben *et al* (1994, 1995) work by (partially) replacing pure random mutation and recombination mechanisms by heuristic ones. Domain independent variable- and value-ordering heuristics from the classical constructive CSP solving methods are adopted. There are two kinds of heuristic, one for selecting the position to change in a chromosome and one for choosing a new value for a selected variable. The heuristic for selecting the position to change chooses that gene i which causes the most severe violation in terms of $\sum_{c_j \in C^i} \chi(s, c_j)$. The heuristic for value selection choose a value that leads to the highest decrease in penalty. Using these problem independent techniques the performance of genetic algorithms (GAs) on CSPs can be highly increased.

Another option is to dynamically refocus the search by changing the objective function f. The basic idea is that the weights are periodically modified by an adaptive mechanism depending on the progress of the search. If in the best individual a constraint is not satisfied, or a variable is instantiated to a value that violates constraints, then the corresponding weight is increased. We have applied this approach successfully for solving CSPs. We observed (Eiben and Ruttkay 1996) that the GA was able to learn constraint weights that were to a large extent independent from the applied genetic operators and initial constraint weights. We showed (Eiben and van der Hauw 1996) that this technique is very powerful and it resulted in a superior graph coloring algorithm. A big advantage of this technique is that it is problem independent.

Coevolutionary constraint satisfaction (Paredis 1994) can also be seen as a particular implementation of adapting the penalty function. Dynamically changing penalties were also applied by Michalewicz and Attia (1994) and Smith and Tate (1993), for solving (continuous) COPs.

If we have chosen to transform the original CSP to a COP (whether or not through a decoder), then we have to take care to enforce the constraints in the allowability condition. This is typically done by either:

(i) *eliminating* newborn individuals if they are not allowable,
(ii) *preserving* allowability, i.e. using special operators that guarantee that allowable parents have allowable children, or
(iii) *repairing* newborn individuals if they are not allowable.

The eliminating approach is generally very inefficient, therefore hardy practicable. Repair algorithms are treated in Section C5.4, while Section C5.5 handles constraint-preserving operators, therefore we omit a detailed discussion here. Let us, however, make a remark on the preserving approach. When choosing the subset of the constraints incorporated in ψ or in f one should keep in mind that the more constraints are represented by f, the less informative the evaluation of the individuals is. For instance, we know that two constraints are violated, but we do not know which ones. Hence, the performance of an EA can be expected to be raised by putting many constraints in ψ and few constraints in f. Nevertheless, this requires genetic operators that maintain many constraints. Thus, the representation issue, i.e. what constraints to keep in the allowability criterion, cannot be treated independently from the operator issue (see also De Jong 1987).

C5.4, C5.5

C5.7.5 Conclusions

In this section we defined three problem classes, FOPs, CSPs, and COPs. In this framework we gave a systematic overview of possibilities to apply EAs for CSPs. Since a CSP has no optimization component it has to be transformed to an FOP or a COP that subsumes it and an EA should be applied to the transformed problem. The FOP option means unconstrained search, thus success of this approach depends on the ability of minimizing penalties. To this end we have sketched two problem independent extensions to a general evolutionary optimizer:

- applying heuristic operators based on classical CSP solving techniques that presumably reduce the level of constraint violation, and
- using adaptive penalty functions based on an updating mechanism of weights, thus allowing the EA to redefine the relative importance of constraints or variables to focus on.

Once we have a COP to be solved by an EA the constraints in the feasibility (allowability) conditions have to be taken care of, while still having to minimize penalties. Solving COPs by EAs has already received a lot of attention: for detailed discussions we refer to other sections of this handbook. Let us note, however, that the EA extensions mentioned for the FOP-based approach are also applicable for improving the performance of an EA working on a COP.

The quoted examples and other successful case studies (e.g. Dozier *et al* 1994, Hao and Dorne 1994) suggest that for many CSPs it is possible to find effective and efficient EAs. However, there is no general recipe for how to handle a CSP by EAs. Our experiments with graph coloring seem to confirm the conclusions of Richardson *et al* (1989) and Michalewicz (1996) that on tough problems the simple penalty function approach is not the best option.

An open research topic is whether one could forge EAs which construct a solution for a CSP, instead of searching in the space of complete instantiations of the variables. Actually, our application of decoders corresponds to deciding in which order the variables are instantiated (nodes are colored). Hence, the decoder technique can be seen as a method to construct different partial or complete instantiations of the variables. Whether it is possible to forge such EAs which operate on partial instantiations directly is an interesting research issue.

Optimal tuning of the EA (e.g. population size or mutation rates) remains an open issue. As the proper selection of these parameters very much depends on the characteristics of the solution space (how many solutions there are, how they are distributed), one can expect guidelines for specific types of CSP for which these characteristics have been investigated. On the basis of the characterization of the fitness landscape one may forecast the effectiveness and efficiency of a genetic operator (Manderick and Spiessens 1994).

We have given just one example of the possible benefits of adaptivity. Besides adapting penalties, EAs could also dynamically adjust parameters based on the evaluation of past experiences. Adaptively modifying operator probabilities (Davis 1989), mutation rates (Bäck 1992), or the population size (Arabas *et al* 1994) has led to interesting results. In addition to these parameters, similar techniques could be used to modify the heuristics, thus the genetic operators, based on earlier performance.

Finally, a hard nut is the question of unsolvable CSPs, since in general an EA cannot conclude for sure that a problem is not solvable. In the particular case of arc inconsistency the results of Bowen and Dozier (1995) are, however, very promising.

References

Arabas J, Michalewicz Z and Mulawka J 1994 GAVaPS—a genetic algorithm with varying population size *Proc. 1st. IEEE Int. Conf. on Evolutionary Computation (Orlando, FL, June 1994)* (Piscataway, NJ: IEEE) pp 306–11

Bäck T 1992 Self-adaptation in genetic algorithms *Proc. 1st European Conference on Artificial Life* ed F J Varela and P Bourgine (Cambridge, MA: MIT Press) pp 263–71

Bowen J and Dozier G 1995 Solving constraint-satisfaction problems using a genetic/systematic search hybrid that realizes when to quit *Proc. 6th Int. Conf. on Genetic Algorithms (Pittsburgh, PA, July 1995)* ed L J Eshelman (San Mateo, CA: Morgan Kaufmann) pp 122–9

Cheeseman P, Kenefsky B and Taylor W M 1991 Where the really hard problems are *Proc. IJCAI-91* (San Mateo, CA: Morgan Kaufmann) pp 331–7

Davis L 1989 Adapting operator probabilities in genetic algorithms *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, June 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 61–9

——1991 *Handbook of Genetic Algorithms* (New York: Van Nostrand Reinhold)

Dechter R 1990 Enhancement schemes for constraint processing: backjumping, learning, and cutset decomposition *Artificial Intell.* **41** 273–312

De Jong K A 1987 On using GAs to search problem spaces *Proc. 2nd Int. Conf. on Genetic Algorithms (Cambridge, MA, 1987)* ed J J Grefenstette (Hillsdale, NJ: Erlbaum) pp 210–6

Dozier G, Bowen J and Bahler D 1994 Solving small and large constraint-satisfaction problems using a heuristic-based microgenetic algorithms *Proc. 1st IEEE World Conf. on Evolutionary Computation (Orlando, FL)* (Piscataway, NJ: IEEE) pp 306–11

Eiben A E, Raue P-E and Ruttkay Zs 1994 Solving constraint-satisfaction problems using genetic algorithms *Proc. 1st IEEE World Conf. on Evolutionary Computation (Orlando, FL)* (Piscataway, NJ: IEEE) pp 542–7

——1995 Constrained problems *Practical Handbook of Genetic Algorithms* ed L Chambers (Boca Raton, FL: Chemical Rubber Company) pp 307–65

Eiben A E and Ruttkay Zs 1996 Self-adaptivity for constraint satisfaction: learning penalty functions *Proc. 3rd IEEE Conf. on Evolutionary Computation* (Piscataway, NJ: IEEE) pp 258–61

Eiben A E and van der Hauw J K 1996 *Graph Coloring with Adaptive Evolutionary Algorithms* Technical Report TR-96-11, Leiden University ftp://ftp.wi.leidenuniv.nl/pub/CS/TechnicalReports/1996/tr96-11.ps.gz

Fox B R and McMahon M B 1991 Genetic operators for sequencing problems *Proc. Workshop on the Foundations of Genetic Algorithms* ed G J E Rawlins (San Mateo, CA: Morgan Kaufmann) pp 284–300

Hao J K and Dorne R 1994 An empirical comparison of two evolutionary methods for satisfiability problems *Proc. 1st. IEEE Int. Conf. on Evolutionary Computation (Orlando, FL, June 1994)* (Piscataway, NJ: IEEE) pp 450–5

Mackworth A K 1977 Concistency in networks of relations *Artificial Intell.* **8** 99–118

Manderick B and Spiessens P 1994 How to select genetic operators for combinatorial optimization problems by analyzing their fitness landscapes *Computational Intelligence: Imitating Life* ed J M Zurada, R J Marks and C J Robinson (Piscataway, NJ: IEEE) pp 170–81

Meseguer P 1989 Constraint-satisfaction problems: an overview *AICOM* **2** 3–17

Michalewicz Z 1995 Genetic algorithms, numerical optimization and constraints *Proc. 6th Int. Conf. on Genetic Algorithms (Pittsburgh, PA, July 1995)* ed L J Eshelman (San Mateo, CA: Morgan Kaufmann) pp 151–8

——1996 *Genetic Algorithms + Data Structures = Evolutionary Computation* 3rd edn (Berlin: Springer)

Michalewicz Z and Attia N 1995 Evolutionary optimization of constrained problems *Proc. 3rd Ann. Conf. on Evolutionary Programming (San Diego, CA, February 1994)* ed A V Sebald and L J Fogel (Singapore: World Scientific) pp 98–108

Michalewicz Z and Michalewicz M 1995 Pro-life versus pro-choice strategies in evolutionary computation techniques *Computational Intelligence: a Dynamic System Perspective* ed M Palaniswami, Y Attikiouzel, R J Marks, D Fogel and T Fukuda (Piscataway, NJ: IEEE) pp 137–51

Minton S, Johnston M D, Philips A and Laird P 1992 Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems *Artificial Intell.* **58** 161–205

Nudel B 1983 Consistent-labeling problems and their algorithms: expected complexities and theory based heuristics *Artificial Intell.* **21** 135–78

Olivier I M, Smith D J and Holland J C R 1987 A study of permutation crossover operators on the travelling salesman problem *Proc. 2nd Int. Conf. on Genetic Algorithms (Cambridge, MA, 1987)* ed J J Grefenstette (Hillsdale, NJ: Erlbaum) pp 224–30

Paredis J 1994 Co-evolutionary constraint satisfaction *Proc. 3rd Parallel Problem Solving from Nature* ed Y Davisor, H-P Schwefel and R Männer (Springer) pp 46–55

Richardson J T, Palmer M R, Liepins G and Hilliard M 1989 Some guidelines for genetic algorithms with penalty functions *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, June 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 191–7

Smith A E and Tate D M 1993 Genetic optimization using a penalty function *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 499–505

Starkweather T, McDaniel S, Mathias K, Whitley D and Whitley Shaner C 1991 A comparison of genetic sequencing operators *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, July 1991)* ed R K Belew and L B Booker (San Mateo, CA: Morgan Kaufmann) pp 69–76

Tsang E 1993 *Foundations of Constraint Satisfaction* (New York: Academic)

C6.1 Niching methods

Samir W Mahfoud

Abstract

Niching methods extend genetic algorithms from optimization to domains such as classification, multimodal function optimization, simulation of complex and adaptive systems, and multiobjective function optimization. Sharing and crowding are two prominent categories of niching methods. Both categories contain algorithms that can successfully locate and maintain multiple solutions within the population of a genetic algorithm.

C6.1.1 Introduction

Niching methods (Mahfoud 1995a) extend *genetic algorithms* (GAs) to domains that require the location and maintenance of multiple solutions. While traditional GAs primarily perform optimization, GAs that incorporate niching methods are more adept at problems in classification and machine learning, multimodal function optimization, *multiobjective function optimization*, and *simulation* of complex and adaptive systems. B1.2, C4.5, F1.9, F1.8

Niching methods can be divided into families or categories, based upon structure and behavior. To date, two of the most successful categories of niching methods are *fitness sharing* (also called *sharing*) and *crowding*. Both categories contain methods that are capable of locating and maintaining multiple solutions within a population, whether those solutions have identical or differing fitnesses.

C6.1.2 Fitness sharing

Fitness sharing, as introduced by Goldberg and Richardson (1987), is a fitness scaling mechanism that alters only the fitness assignment stage of a GA. Sharing can be used in combination with other scaling mechanisms, but should be the last one applied, just prior to selection.

From a multimodal function maximization perspective, the idea behind sharing is as follows. If similar individuals are required to share payoff or fitness, then the number of individuals that can reside in any one portion of the *fitness landscape* is limited by the fitness of that portion of the landscape. Sharing results in individuals being allocated to optimal regions of the fitness landscape. The number of individuals residing near any peak will theoretically be proportional to the height of that peak. B2.7

Sharing works by derating each population element's fitness by an amount related to the number of similar individuals in the population. Specifically, an element's *shared fitness*, F', is equal to its prior fitness F divided by its *niche count*. An individual's niche count is the sum of *sharing function* (sh) values between itself and each individual in the population (including itself). The shared fitness of a population element i is given by the following equation:

$$F'(i) = \frac{F(i)}{\sum_{j=1}^{\mu} \mathrm{sh}(d(i, j))}. \tag{C6.1.1}$$

The sharing function is a function of the distance d between two population elements; it returns a '1' if the elements are identical, a '0' if they cross some threshold of dissimilarity, and an intermediate value for intermediate levels of dissimilarity. The threshold of dissimilarity is specified by a constant, σ_{share}; if

the distance between two population elements is greater than or equal to σ_{share}, they do not affect each other's shared fitness. A common sharing function is

$$\text{sh}(d) = \begin{cases} 1 - (d/\sigma_{\text{share}})^{\alpha} & \text{if } d < \sigma_{\text{share}} \\ 0 & \text{otherwise} \end{cases} \tag{C6.1.2}$$

where α is a constant that regulates the shape of the sharing function.

While nature distinguishes its niches based upon phenotype, niching GAs can employ either *genotypic* or *phenotypic* distance measures. The appropriate choice depends upon the problem being solved.

C6.1.2.1 Genotypic sharing

In genotypic sharing, the distance function d is simply the Hamming distance between two strings. (The Hamming distance is the number of bits that do *not* match when comparing two strings.) Genotypic sharing is generally employed by default, as a last resort, when no phenotype is available to the user.

C6.1.2.2 Phenotypic sharing

In phenotypic sharing, the distance function d is defined using problem-specific knowledge of the phenotype. Given a function optimization problem containing k variables, the most common choice for a phenotypic distance function is Euclidean distance. Given a classification problem, the phenotypic distance between two classification rules can be defined based upon the examples to which they both apply.

C6.1.2.3 Parameters and extensions

Typically, α is set to unity, and σ_{share} is set to a value small enough to allow discrimination between desired peaks. For instance, given a one-dimensional function containing two peaks that are two units apart, a σ_{share} of 1 is ideal: since each peak extends its reach for $\sigma_{\text{share}} = 1$ unit in each direction, the reaches of the peaks will touch but not overlap. Deb (1989) gives more details for setting σ_{share}.

Population size can be set roughly as a multiple of the number of peaks the user wishes to locate (Mahfoud 1995a, b). Sharing is best run for few generations, perhaps some multiple of $\log \mu$. This rough heuristic comes from shortening the expected convergence time for a GA that uses fitness-proportionate selection (Goldberg and Deb 1991). A GA under sharing will not converge population elements atop the peaks it locates. One way of obtaining such convergence is to run a hillclimbing algorithm after the GA. E1.1

Sharing can be implemented using any selection method, but the choice of method may either increase or decrease the stability of the algorithm. *Fitness-proportionate* selection with stochastic universal sampling (Baker 1987) is one of the more stable options. *Tournament* selection is another possibility, but special provisions must be made to promote stability. Oei *et al* (1991) propose a technique for combining sharing with binary tournament selection. This technique, *tournament selection with continuously updated sharing*, calculates shared fitnesses with respect to the new population as it is being filled. C2.2 C2.3

The main drawback to using sharing is the additional time required to cycle through the population to compute shared fitnesses. Several authors have suggested calculating shared fitnesses from fixed-sized samples of the population (Goldberg and Richardson 1987, Oei *et al* 1991). Clustering is another potential remedy. Yin and Germay (1993) propose that a clustering algorithm be implemented prior to sharing, in order to divide the population into niches. Each individual subsequently shares only with the individuals in its niche. As far as GA time complexity is concerned, in real-world problems, a function evaluation requires much more time than a comparison; most GAs perform only $O(\mu)$ function evaluations each generation.

C6.1.3 Crowding

Crowding techniques (De Jong 1975) insert new elements into the population by replacing similar elements. To determine similarity, crowding methods, like sharing methods, utilize a distance measure, either genotypic or phenotypic. Crowding methods tend to spread individuals among the most prominent peaks of the search space. Unlike sharing methods, crowding methods do not allocate elements proportional to peak fitness. Instead, the number of individuals congregating about a peak is largely determined by the size of that peak's basin of attraction under crossover.

By replacing similar elements, crowding methods strive to maintain the preexisting diversity of a population. However, replacement errors may prevent some crowding methods from maintaining individuals in the vicinity of desired peaks. The *deterministic crowding* algorithm (Mahfoud 1992, 1995a) is designed to minimize the number of replacement errors, and thus allow effective niching.

Deterministic crowding works as follows. First it groups all population elements into $\mu/2$ pairs. Then it crosses all pairs and mutates the offspring. Each offspring competes against one of the parents that produced it. For each pair of offspring, two sets of parent–child tournaments are possible. Deterministic crowding holds the set of tournaments that forces the most similar elements to compete.

The pseudocode for deterministic crowding is as follows:

Input: g—number of generations to run, μ—population size
Output: $P(g)$—the final population

$P(0) \leftarrow$ initialize()
for $t \leftarrow 1$ **to** g **do**
  $P(t) \leftarrow$ shuffle($P(t-1)$)
  for $i \leftarrow 0$ **to** $\mu/2 - 1$ **do**
    $p_1 \leftarrow \boldsymbol{a}_{2i+1}(t)$
    $p_2 \leftarrow \boldsymbol{a}_{2i+2}(t)$
    $\{c_1, c_2\} \leftarrow$ recombine(p_1, p_2)
    $c_1' \leftarrow$ mutate(c_1)
    $c_2' \leftarrow$ mutate(c_2)
    if $[d(p_1, c_1') + d(p_2, c_2')] \leq [d(p_1, c_2') + d(p_2, c_1')]$ **then**
      if $F(c_1') > F(p_1)$ **then** $\boldsymbol{a}_{2i+1}(t) \leftarrow c_1'$ **fi**
      if $F(c_2') > F(p_2)$ **then** $\boldsymbol{a}_{2i+2}(t) \leftarrow c_2'$ **fi**
    else
      if $F(c_2') > F(p_1)$ **then** $\boldsymbol{a}_{2i+1}(t) \leftarrow c_2'$ **fi**
      if $F(c_1') > F(p_2)$ **then** $\boldsymbol{a}_{2i+2}(t) \leftarrow c_1'$ **fi**
    fi
  od
od

Deterministic crowding requires the user only to select a population size μ and a stopping criterion. As a general rule of thumb, the more final solutions a user desires, the higher μ should be. The user can stop a run after either a fixed number of generations g (of the same order as μ) or when the rate of improvement of the population approaches zero. Full crossover should be employed (with probability 1.0) since deterministic crowding only discards solutions after better ones become available, thus alleviating the problem of crossover disruption.

Two crowding methods similar in operation and behavior to deterministic crowding have been proposed (Cedeño *et al* 1994, Harik 1995). Cedeño *et al* suggest utilizing phenotypic crossover and mutation operators (i.e. *specialized* operators), in addition to phenotypic sharing; this results in further reduction of replacement error.

C6.1.4 Theory

Much of the theory underlying sharing, crowding, and other niching methods is currently under development. However, a number of theoretical results exist, and a few areas of theoretical research have already been defined by previous authors. The characterization of hard problems is one area of theory. For niching methods, the number of optima the user wishes to locate, in conjunction with the number of optima present, largely determines the difficulty of a problem. A secondary factor is the degree to which extraneous optima lead away from desired optima.

Analyzing the distribution of solutions among optima for particular algorithms forms another area of theory. Other important areas of theory are calculating expected drift or disappearance times for desired solutions; population sizing; setting parameters such as operator probabilities and σ_{share} (for sharing); and improving the designs of niching genetic algorithms. For an extensive discussion of niching methods and their underlying theory, consult the article by Mahfoud (1995a).

References

Baker J E 1987 Reducing bias and inefficiency in the selection algorithm *Proc. Int. Conf. on Genetic Algorithms (Cambridge, MA)* ed J J Grefenstette (Hillsdale, NJ: Lawrence Erlbaum Associates) pp 14–21

Cedeño W, Vemuri V R and Slezak T 1994 Multiniche crowding in genetic algorithms and its application to the assembly of DNA restriction-fragments *Evolutionary Comput.* **2** 321–45

Deb K 1989 *Genetic Algorithms in Multimodal Function Optimization* Masters Thesis; TCGA Report 89002, University of Alabama, The Clearinghouse for Genetic Algorithms

De Jong K A 1975 *An Analysis of the Behavior of a Class of Genetic Adaptive Systems* Doctoral Dissertation, University of Michigan *Dissertation Abstracts Int.* **36** 5140B (University Microfilms 76-9381)

Goldberg D E and Deb K 1991 A comparative analysis of selection schemes used in genetic algorithms *Foundations of Genetic Algorithms* ed G J E Rawlins (San Mateo, CA: Morgan Kaufmann) pp 69–93

Goldberg D E and Richardson J 1987 Genetic algorithms with sharing for multimodal function optimization *Proc. 2nd Int. Conf. on Genetic Algorithms (Cambridge, MA)* ed J J Grefenstette (Hillsdale, NJ: Lawrence Erlbaum Associates) pp 41–9

Harik G 1995 Finding multimodal solutions using restricted tournament selection *Proc. 6th Int. Conf. on Genetic Algorithms (Pittsburgh, July 1995)* ed L J Eshelman (San Mateo, CA: Morgan Kaufmann) pp 24–31

Mahfoud S W 1992 Crowding and preselection revisited *Parallel Problem Solving From Nature* vol 2, ed R Männer and B Manderick (Amsterdam: Elsevier) pp 27–36

——1995a *Niching Methods for Genetic Algorithms* Doctoral Dissertation and IlliGAL Report 95001, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory) *Dissertation Abstracts Int.* (University Microfilms 9543663)

——1995b Population size and genetic drift in fitness sharing *Foundations of Genetic Algorithms* vol 3, ed L D Whitley and M D Vose (San Francisco, CA: Morgan Kaufmann) pp 185–223

Oei C K, Goldberg D E and Chang S J 1991 *Tournament Selection, Niching, and the Preservation of Diversity* (IlliGAL Report 91011) University of Illinois, Illinois Genetic Algorithms Laboratory

Yin X and Germay N 1993 A fast genetic algorithm with sharing scheme using cluster analysis methods in multimodal function optimization *Artificial Neural Nets and Genetic Algorithms: Proc. Int. Conf. (Innsbruck)* ed R F Albrecht, C R Reeves and N C Steele (Berlin: Springer) pp 450–7

C6.2 Speciation methods

Kalyanmoy Deb (C6.2.1–C6.2.4) *and William M Spears* (C6.2.5, C6.2.6)

Abstract

In nature, a species is defined as a collection of phenotypically similar individuals. Many biologists believe that individuals in a sexually reproductive species can be created and maintained by allowing restrictive mating only among individuals from the same species. The connection between the formation of multiple species in nature and in search and optimization problems lies in solving multimodal problems, where the objective is not only to find one optimal solution, but to find a number of optimal solutions. In those problems, each optimal solution may be assumed to constitute a species. Since evolutionary algorithms work with a population of solutions, the concept of natural speciation techniques can be implemented to allow formation of multiple subpopulations, each focusing its search for one optimal solution. This way, multiple optimal solutions can be discovered simultaneously. In this section, a number of speciation techniques are discussed.

C6.2.1 Introduction

Kalyanmoy Deb

Despite some controversy, most biologists agree that a species is a collection of individuals which resemble each other more closely than they resemble individuals of another species (Eldredge 1989). It is also clear that the reproductive process of the sexually reproducing organisms causes individuals to resemble their parents, thereby maintaining a phenotypic similarity among individuals of the community or the *species*. Thus, there is a strong correlation among the reproductively coherent individuals and a phenotypically similar cluster of individuals. Since in evolutionary algorithms a population of solutions is used, artificial species of phenotypically similar solutions can be formed and maintained in the population by restricting their mating to that with similar individuals. Before we outline how to form and maintain multiple species in a population, let us discuss why it could be necessary to form species in the applications of evolutionary algorithms.

In Section C6.1, we saw that multiple optimal solutions in a multimodal optimization problem can be found simultaneously by forming artificial niches (subpopulations) in the population. Each niche can be considered to represent a peak (in the spirit of maximization problems). To capture a number of peaks simultaneously and maintain them for many generations, a niching method is used. Niching helps to emphasize and maintain solutions around multiple optima. However, in niching, the main emphasis is devoted to distributing the population members across different peaks. Thus, the niching technique cannot quite focus its search on each peak and find the exact optimal solutions efficiently. This is because some of the search effort is wasted in the recombination of interpeak solutions, which, in turn, may produce some *lethal* solutions representing none of the peaks. A speciation method used in evolutionary computation (EC) studies, on the other hand, restricts mating to that among like solutions (likeness can be defined phenotypically or genotypically) and discourages mating among solutions of different peaks. If the likeness is defined properly, two parent solutions chosen for mating are likely to represent the same peak. Thus, when like individuals mate with each other, the created children solutions are also similar to the

C6.1

parent solutions and are likely to be members of the same peak. This way, the restriction of mating to that among like solutions may reduce the creation of lethal solutions (which represent none of the peaks). This may allow the search to concentrate on each peak and help find the best or near-best optimum solution efficiently. However, in order to apply the speciation technique properly, solutions representing each peak must first be found. Thus, the speciation technique cannot be used independently. In the presence of both niching and speciation, niching finds and maintains subpopulation of solutions around multiple optima and the speciation technique allows us to make an inherent parallel search in each optimum to find multiple optimal solutions simultaneously.

Among the evolutionary algorithms, a number of speciation methods have been suggested and implemented in *genetic algorithms* (GAs). Of the earlier works related to mating restriction in GAs, B1.2 Hollstien's (1971) inbreeding scheme where mating was allowed between similar individuals in his simulation of animal husbandry problems, Booker's (1982) taxon–exemplar scheme for restrictive mating in his simulation of learning pattern classes, Holland's suggestion of a tag–template scheme (Goldberg 1989), Sannier and Goodman's (1987) restrictive mating in forming separate coherent groups in a population, Deb's (1989) phenotypic and genotypic mating restriction schemes, and Spears's (1994) and Perry's (1984) speciation using tag bits are a few studies. In the following, we discuss some of the above speciation methods in more detail.

C6.2.2 Booker's taxon–exemplar scheme

Kalyanmoy Deb

Booker (1982) used taxons and exemplars in his *learning algorithm* to reduce the formation of lethal B1.5.2 individuals. He defined a taxon as a string (constructed over the three-letter alphabet {0, 1, #}, with a # matching a 0 or a 1). The population is initialized with taxon strings. In his *restricted mating policy*, he wanted to restrict mating among similar taxon strings, which were identified by calculating a match score of the taxon strings with a given exemplar binary string. He allowed partial match scores depending on the matching of the taxon and the exemplar. For the following two taxon strings and the exemplar string, the first taxon matches the exemplar completely. The second taxon matches the exemplar partially (in first, third, and fourth positions):

Taxon	Exemplar
(1 # 0 0 #)	(1 0 0 0 0).
(# 1 # 0 1)	

If the taxon completely matches the exemplar, a score is assigned as the sum of the string length and the number of #s in the taxon. The partial credit is also assigned based on the number of correct matches and the number of #s in the taxon. In order to implement the restrictive mating policy, he chose parent taxon strings from a sample subpopulation determined based on the available matching taxon strings in the population. If a specified number of matching taxon strings are available in the population, parent strings are chosen uniformly at random from all the matching taxon strings. Otherwise, parent strings are chosen according to a probability distribution calculated based on the match score of the taxon strings. In a number of pattern discovery problems an improved performance is observed with the restricted mating policy.

After the patterns were discovered, Booker extended his above scheme to classify the discovered patterns using a modified string as follows:

Taxon		Tag
(1 # 0 0 #)	:	(1 0 0 0 0).

In addition to the taxon string, a tag string is introduced to classify the discovered taxon strings (or patterns). The taxon strings matching a particular tag string were considered to be in the same class. A similar match score was used, except that this time the matching was performed with the taxon and tag strings. As discussed elsewhere (Goldberg 1989), there is one difficulty with the above tag–taxon scheme. The tag string must be of the same length as the taxon string. This increases the complexity of the classification problem, whereas the same concept can be implemented with shorter tag and template strings, as suggested by Holland; a brief description of this is given by Goldberg (1989).

C6.2.3 The tag–template method

Kalyanmoy Deb

In addition to the functional string (the taxon string in Booker's pattern classification problem), a template and a tag string are introduced. The template string is constructed from the three-letter alphabet (1, 0, and #) as before, but the tag string is a binary string of the same length as the template string. A typical string with the tag and template strings would look like the following:

Template		Tag		Functional string
(#01)	:	(100)	:	(1011001101).

The size of tag and template strings depends on the number of desired solutions. A simple calculation shows that if q different optimal solutions (peaks) are to be found, the minimum string length for the tag and template is $\lceil \log_2 q \rceil$ (Deb 1989). The tag and template strings are created at random in the initial population along with the functional string. These two strings do not affect the fitness of the functional string. However, they are affected by the crossover and the mutation operators, as well. For the template string, the mutation operator must be modified to operate on a three-allele string. The purpose of these strings is to restrict mating. Before crossing a pair of individual strings, their tag and template strings are *matched*. If the match score exceeds a threshold value, the crossover is performed between the two strings as usual; otherwise some other string pair is tested for a possible mating. In this process, the tag and template strings corresponding to the good individuals in early populations are emphasized and an artificial tag is set for solutions in each peak. Later on, since crossing over is only performed between the matched strings, only similar strings (or strings from the same peak) tend to participate in crossover.

Although neither Holland nor Goldberg simulated this speciation method, Deb (1989) (with assistance from David Goldberg) implemented this scheme and applied this technique in solving multimodal test problems. In both cases, GAs with the tag–template scheme performed better than GAs without it.

C6.2.4 Phenotypic and genotypic mating restriction

Kalyanmoy Deb

Deb (1989) has developed two mating restriction schemes based on the phenotypic and genotypic distance between mating individuals. The mating restriction schemes are straightforward. In order to choose a mate for an individual, their distance (in phenotypic mating restriction the Euclidean distance and in genotypic mating restriction the Hamming distance) is computed. If the distance is closer than a parameter σ_{mating}, they participate in the crossover operation; otherwise another individual is chosen at random and their distance is computed. This process is continued until a suitable mate is found or all population members are exhausted, in which case a random individual is chosen as a mate. Deb has implemented both the above mating restriction schemes with a single-point crossover and applied them to solve a number of multimodal test problems. Although, in all his simulations, the parameter σ_{mating} was kept the same as the parameter σ_{share} used in the niching methods, other values of σ_{mating} may also be chosen. It is worthwhile to mention that niching with the σ_{share} parameter is implemented in the selection operator and the mating restriction with the σ_{mating} parameter is implemented in the crossover operator. GAs with niching and mating restriction were found to better distribute the population across the peaks than GAs with sharing alone. Here, we present simulation results for the phenotypic mating restriction scheme adopted in that study. In solving the single-variable, five-peaked function in the interval $0 \leq x \leq 1$

$$\text{maximize} \qquad 2^{-2((x-0.1)/0.8)^2} \sin^6(5\pi x)$$

with $\sigma_{\text{share}} = \sigma_{\text{mating}} = 0.1$, 100 population members after 200 generations without and with phenotypic mating restriction are shown in figure C6.2.1. Stochastic remainder roulette wheel selection and single-point crossover operators are used. The crossover and mutation probabilities are kept as 0.9 and 0.0, respectively. The figures show that, with the mating restriction scheme (the right-hand panel), the number of lethal (nonpeak) individuals has been significantly decreased. This study also implemented a genotypic mating restriction scheme and similar results were obtained. Some guidelines in choosing the sharing and mating restriction parameters are outlined elsewhere (Deb 1989, Deb and Goldberg 1989).

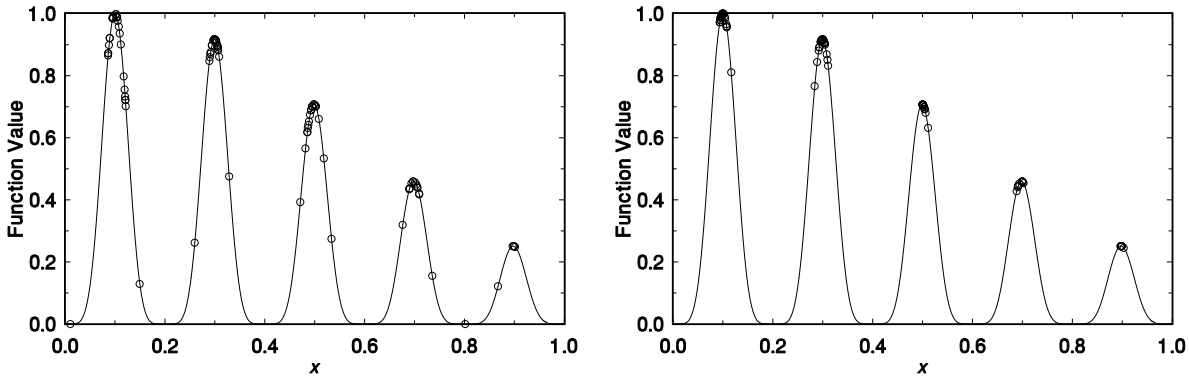


Figure C6.2.1. The distribution of 100 solutions without (left) and with (right) a mating restriction scheme.

C6.2.5 Speciation using tag bits

William M Spears

Another method for identifying species is via the use of tag bits, which are appended to every individual. Each species corresponds to a particular setting of these bits. Suppose there are k different sets of tag bit values at a particular generation of the evolutionary algorithm (EA). Denote these sets as $\{S_0, \ldots, S_{k-1}\}$. The sets are numbered arbitrarily. Each individual belongs to one S_i and all individuals in a particular S_i have the same tag bit values. For example, suppose there is only one tag bit and that some individuals exist with a tag bit value zero and that the remainder exist with tag bit value one. Then (arbitrarily) assign the former set of individuals to S_0 and the latter set to S_1. Let $\| \ \|$ denote the cardinality of the sets.

Spears (1994) uses the tag bits to restrict mating and to perform fitness sharing. With sharing, the perceived fitness, F_i, is a normalization of the objective fitness f_i:

$$F_i = \frac{f_i}{\|S_j\|} \qquad i \in S_j$$

where $\|S_j\|$ is the size of the species that individual i is in.

The average fitness of the population, $\bar{F}$, becomes

$$\bar{F} = \frac{\sum_{i \in S_0} (f_i / \|S_0\|) + \ldots + \sum_{i \in S_{k-1}} (f_i / \|S_{k-1}\|)}{\|S_0\| + \ldots + \|S_{k-1}\|}$$

which is just

$$\bar{F} = \frac{\sum_{i \in S_0} (f_i / \|S_0\|) + \ldots + \sum_{i \in S_{k-1}} (f_i / \|S_{k-1}\|)}{N}$$

since the species sizes have to total N (recall that no individual can lie in more than one species). The expected number of offspring for an individual is now $F_i / \bar{F}$.

Restricted mating is performed by only allowing recombination to occur between individuals with the same tag bit values. Mutation can flip all bits, including the tag bits, thus allowing individuals to change labels. Experimental results, as well as some modifications to the above mechanism can be found in the article by Spears (1994). Code for the algorithm can be found at http://www.aic.nrl.navy.mil/~spears.

Perry's thesis work (Perry 1984) with speciation is extremely similar to the above technique. Perry includes both species and environmental regions in an EA. Species are identified via tag bits and an environmental region is similar to an EA population. Recombination within an environment can occur only on individuals with the same tag bit values. Mutation is allowed to change tag bits, in order to introduce new species. The additional use of a 'migration' operator, which moves individuals from one environment to another, does not have an analog in the work of Spears (1994).

Perry gives an example of two species in an environment—fitness proportional selection is performed, and the average fitness of an environment is

$$\bar{f} = \frac{\sum_{i \in S_0} f_i + \sum_{i \in S_1} f_i}{\|S_0\| + \|S_1\|}$$

or

$$\bar{f} = \frac{\sum_{i \in S_0} f_i + \sum_{i \in S_1} f_i}{N}$$

where N is the population size of the environmental niche. The expected number of offspring is $f_i / \bar{f}$. One can see that the main difference between the two methods is the use of sharing in the computation of fitness in the work of Spears (1994). Thus it is not surprising that in many of Perry's experimental runs one particular species would eventually dominate an environmental niche (however, it should be noted that in the work of Perry (1984) the domination of an environment by a species was not undesirable behavior).

The use of tag bits makes restricted mating and fitness sharing more efficient because distance comparisons do not have to be computed. Interestingly, it is also possible to make Goldberg's implementation of sharing more efficient by sampling (Goldberg *et al* 1992). In other words the distance of each individual from the rest is estimated by using a subset of the remaining individuals.

C6.2.6 Relationship with parallel algorithms

William M Spears

Clearly this work has similarities to the EA research performed on *parallel architectures*. In a parallel EA, a topology is imposed on the EA population, resulting in species. However, there are some important differences between the parallel approaches and the sequential approach. For example, with the fitness sharing approaches the fitness of an individual and the species size are dynamic, based on the other individuals (and species). This concentrates effort on more promising peaks, while still maintaining individuals in other areas of the search space. This is typically not true for parallel EAs implemented on MIMD or SIMD architectures. When using a MIMD architecture, species are dedicated to particular processors and the species remain a constant size. In SIMD implementations, one or two individuals reside on a processor, and species are formed by defining overlapping neighborhoods. However, due to the overlap, one particular species will eventually take over the whole population.

C6.3.1, C6.4.2

References

Booker L B 1982 *Intelligent Behavior as an Adaptation to the Task Environment* Doctoral Dissertation, University of Michigan; *Dissertation Abstracts Int.* **43** 469B

Deb K 1989 *Genetic Algorithms in Multimodal Function Optimization* Master's Thesis, University of Alabama; TCGA Report 89002

Deb K and Goldberg D E 1989 An investigation of niche and species formation in genetic function optimization *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 42–50

Eldredge N 1989 *Macro-evolutionary Dynamics: Species, Niches and Adaptive Peaks* (New York: McGraw-Hill)

Goldberg D E 1989 *Genetic Algorithms in Search, Optimization, and Machine Learning* (Reading, MA: Addision-Wesley)

Goldberg D E and Richardson J 1987 Genetic algorithms with sharing for multimodal function optimization *Proc. 2nd Int. Conf. on Genetic Algorithms (Cambridge, MA, 1987)* ed J J Grefenstette (Hillsdale, NJ: Erlbaum) pp 41–9

Goldberg D E, Deb K and Horn J 1992 Massive multimodality, deception, and genetic algorithms *Proc. Parallel Problem Solving from Nature Conf.* (Amsterdam: North-Holland) pp 37–46

Hollstien R B 1971 *Artificial Genetic Adaptation in Computer Control Systems* Doctoral Dissertation, University of Michigan; *Dissertation Abstracts Int.* **32** 1510B

Perry Z A 1984 *Experimental Study of Speciation in Ecological Niche Theory using Genetic Algorithms* Doctoral Dissertation, University of Michigan; *Dissertation Abstracts Int.* **45** 3870B

Sannier A V and Goodman E D 1987 Genetic learning procedures in distributed environments *Proc. 2nd Int. Conf. on Genetic Algorithms (Cambridge, MA, 1987)* ed J J Grefenstette (Hillsdale, NJ: Erlbaum) pp 162–9

Spears W M 1994 Simple subpopulation schemes *Proc. Conf. on Evolutionary Programming* (Singapore: World Scientific) pp 296–307

C6.3 Island (migration) models: evolutionary algorithms based on punctuated equilibria

W N Martin, Jens Lienig and James P Cohoon

Abstract

The island model genetic algorithm shows promise as a superior formulation based on considerations from theories of natural evolution and from the efficiencies of coarse-grained parallel computer architectures. The theory of *punctuated equilibria* calls for the population to be partitioned into several distinct subpopulations. These subpopulations have extensive periods of isolated evolution, i.e. computation, occasionally interspersed with migration, i.e. communication. It is precisely for this sort of process with its alternating phases of extended computation and limited communication that message-passing multiprocessors (implementing coarse-grained parallelism) are best suited. We validate this promise of the island model and illustrate the effects of varying configuration attributes through experiments with a difficult very large-scale integration (VLSI) design problem.

C6.3.1 Parallelization

Research to develop *parallel implementations* of algorithms has a long history (Slotnick *et al* 1962, Barnes *et al* 1968, Wulf and Bell 1972) across many disparate application areas. The majority of this research has been motivated by the desire to reduce the overall time to completion of a task by distributing the work implied by a given algorithm to processing elements working in parallel. More recently some researchers have conjectured that some parallelizations of a task improve the quality of solution obtained for a given overall amount of work, e.g. emergent computation (Forrest 1991), and some even suggest that considering parallelization may lead to fundamentally new modes of thought (Bailey 1992). Note that the benefits of this latter kind of parallelization depend only on concurrency, i.e. the logical temporal independence, of operations and thus they can also be obtained via sequential simulations of parallel formulations. E2.1

The more prevalent motivation for parallelization, i.e. reducing time to completion, depends on the specifics of the architecture executing the parallelized algorithm. Very early on, it was recognized that different parallel hardware made possible different categories of parallelization based on the *granularity* of the operations performed in parallel. Typically these categories are referred to as *fine-grained*, *medium-grained*, and *coarse-grained* parallelization. At the extremes of this spectrum, fine-grained (or small-grained) parallelism means that only short computation sequences are performed between synchronizations, while coarse-grained (or large-grained) parallelism means that extended computation sequences are performed between synchronizations. SIMD (single-instruction, multiple-data) architectures are most appropriate for fine-grained parallelism (Fung 1976), while distributed-memory message-passing architectures are most appropriate for coarse-grained parallelism (Seitz 1985).

Two of the earliest parallelizations of a *genetic algorithm* (GA) were based on a distributed-memory message-passing architecture (Tanese 1987, Pettey *et al* 1987; also see Grosso (1985) for an early serial simulation of a concurrent formulation). The resulting parallelization was coarse grained in that the overall population of the GA was broken into a relatively small number of subpopulations. Each processing element in the architecture was assigned an entire subpopulation and executed a rather standard GA on its subpopulation. B1.2

In the same time frame, it was noted that a theory concerning speciation and stasis in populations of living organisms, called *punctuated equilibria*, provided evidence that in natural systems this kind of *parallelization* of evolution had an emergent property of bursts of rapid evolutionary *progress*. The resulting parallel GA was shown to have this property on several applications (Cohoon *et al* 1987).

Each of the above systems are examples of what has come to be called *island model* parallel genetic algorithms (Gordon *et al* 1992, Adamidis 1994). In the next section we discuss theories of natural evolution as they support and motivate island model formulations. We then discuss the important aspects, parameters, and attributes of systems built on this model. Finally, we present results of one such system on a difficult very large-scale integration (VLSI) design problem.

C6.3.2 Theories of natural evolution

In what has been called the *modern synthesis* (Huxley 1942), the fields of biological evolution and genetics began to be merged. A major development in this synthesis was Sewall Wright's (1932) conceptualization of the *adaptive landscape*. The original conceptualization proposes an underlying space (two-dimensional for discussion purposes) of possible genetic combinations. At each point in that space an *adaptive value* is determined and specified as a scalar quantity. The surface thus specified is referred to as the *adaptive landscape*. A population of organisms can be mapped to the landscape by taking each member of the population, determining the point in the underlying space that its genetic code specifies, and marking the associated surface point. The figure used repeatedly by Wright shows the adaptive landscape as a standard topographic map with contour lines of equal adaptive value instead of altitude. The + symbols indicate local maxima. A population—in two demes—is then depicted by two shaded regions overlaid on the map.

B2.7

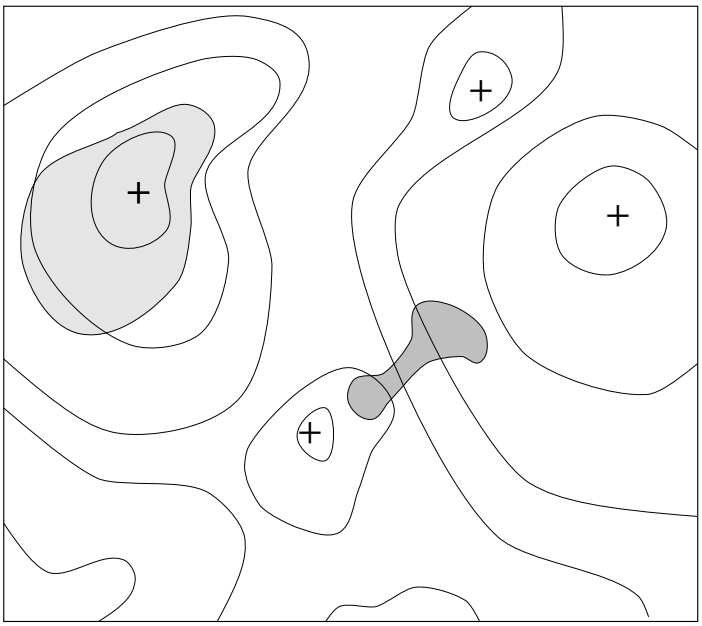


Figure C6.3.1. An adaptive landscape according to Wright, with a sample population—in two demes (shaded areas).

There are several reasons why we used the word *conceptualization* in the previous paragraph. First and foremost, it is not clear what the topology of the underlying space should be. Wright (1932) considers initially the individual gene sequences and connects genetic codes that are 'one remove' from each other, implying that the space is actually an undirected connected graph. He then turns immediately to a continuous space with each gene locus specifying a dimension and with units along each dimension being the possible allelomorphs at the given locus. Specifying the underlying space to be an multidimensional Euclidean space determines the topology. However, if one is to attempt to make inferences from the character of the adaptive landscape (Radcliffe 1991), the ordering of the units along the various dimensions is crucial. With arbitrary orderings the metric notions of *nearby* and *distant* have no clear-cut meaning;

similar ambiguities occur in many discrete optimization problems. For instance, given two tours in a traveling salesperson problem, what is the proper measure of their closeness?

The concept of the adaptive landscape has had a powerful effect on both microevolutionary and macroevolutionary theory, as well as providing a fundamental basis for considering genetic algorithms as function optimizers. As Wright states (1932):

> The problem of evolution as I see it is that of a mechanism by which the species may continually find its way from lower to higher peaks in such a field. In order that this may occur, there must be some trial and error mechanism on a grand scale ...

Wright also used the adaptive landscape concept to explain his mechanism, the *shifting balance theory*. In the shifting balance theory the ability for a species to 'search' and not be forced to remain at lower adaptive peaks by strong selection pressure is provided through a population structure that allows the species to take advantage of ecological opportunities. The population structure is based upon *demes*, as Wright describes (1964):

> Most species contain numerous small, random breeding local populations (demes) that are sufficiently isolated (if only by distance) to permit differentiation ...

Wright conceives the shifting balance to be a microevolutionary mechanism, that is, a mechanism for evolution within a species. For him the emergence of a new species is a corollary to the general operation and progress of the shifting balance. Eldredge and Gould (1972) have contended that macroevolutionary mechanisms are important and see the emergence of a new species as being associated very often with extremely rapid evolutionary development of diverse organisms. As Eldredge states (1989):

> Other authors have gone further, suggesting that SMRS [SMRS denotes *specific mate recognition system*, the disruption of which is presumed to cause reproductive isolation] disruption actually may *induce* [his emphasis] economic adaptive change, i.e., rather than merely occur in concert with it, ... [Eldredge and Gould] have argued that small populations near the periphery of the range of an ancestral population may be ideally suited to rapid adaptive change following the onset of reproductive isolation. ... Thus SMRS disruption under such conditions may readily be imagined to act as a 'release,' or 'trigger' to further adaptive change the better to fit the particular ecological conditions at the periphery of the parental species's range.

The island model GA formulation by Cohoon *et al* (1987, 1991a) was strongly influenced by this theory of punctuated equilibria (Eldredge and Gould 1972), so they dubbed the developed system the *genetic algorithm with punctuated equilibria* (GAPE). In general, the important aspect of the Eldredge–Gould theory is that one should look to small disjoint populations, i.e. peripheral isolates, for extremely rapid evolutionary change.

For the analogy to discrete optimization problems, the peripheral isolates are the semiindependent subpopulations and the rapid evolutionary change is indicative of extensive search of the solution domain. Thus, we contend that the island model genetic algorithm is rightly considered to be based on a population structure that involves subpopulations which have their isolated evolution occasionally punctuated by interpopulation communication (Cohoon *et al* 1991b). To relate these processes to Holland's terms (1975), the *exploration* needed in GAs arises from the infusion of migrants, i.e. individuals from neighboring subpopulations, and the *exploitation* arises from the isolated evolution. It is this alternation between phases of communication and computation that holds the promise for island model GAs to be more than just hardware accelerators for the evolutionary process. In the next section the major aspects of such island models will be delineated.

C6.3.3 The island model

The basic model begins with the islands—the *demes* (Wright 1964) or the *peripheral isolates* (Eldredge and Gould 1972). Here the islands will be referred to as *subpopulations*. It is important to note again that while one motivation in parallelization would demand that each subpopulation be assigned to its own processing element, the islands are really a logical structure and can be implemented efficiently on many different architectures. For this reason we will refer to each subpopulation being assigned to a process, leaving open the issue of how that process is executed.

The island model will consider there to be an overall population $\mathcal{P}$ of $M = |\mathcal{P}|$ individuals that is partitioned into N subpopulations $\{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_N\}$. For an even partition each subpopulation has $\mu = M/N$ individuals, but for generality we can have $|\mathcal{P}_i| = \mu_i$ so that each subpopulation might have a distinct size. For standard GAs the selection of M can be problematic and for island model GAs this decision is compounded by the necessity to select N (and thereby μ). In practice, the decisions often need to be made in the opposite order; that is, μ_i is crucial to the dynamics of the trajectory of evolution for $\mathcal{P}_i$ and is heavily problem dependent. We believe that for specific problems there is a threshold size, below which poor results are obtained (as we will show in section C6.3.5). Further, we believe that island model GAs are less sensitive to the choice of μ_i, as long as it is above the threshold for the problem instance. With μ_i decided, the selection of N (and thereby M) is often based on the available parallel architecture.

Given N, the next decision is the subpopulation interconnection. This is generally referred to as the *communication topology*, in that the island model presumes migration, i.e. intersubpopulation communication. The $\mathcal{P}_i$ are considered to be the vertices of a graph (usually undirected or at least symmetric) with each edge specifying a communication link between the incident vertices. These links are often taken to correspond to actual communication links between the processing elements assigned to the subpopulations. In any case, the communication topology is almost always considered to be static.

Given the ability for two subpopulation processes to communicate, the magnitude and frequency of that communication must be determined. Note that if one allows zero to be a possible magnitude then the communication topology and magnitudes can be specified by a matrix S, where S_{ij} is the number of individuals sent from $\mathcal{P}_i$ to $\mathcal{P}_j$. $S_{ij} = 0$ indicates no communication edge.

As was mentioned in section C6.3.2, the migration pattern is important to the overall evolutionary trajectory. The migration pattern is determined by the degree of connectivity in the communication topology, the magnitude of communication, and the frequency of communication. These parameters determine the amount of isolation and interaction among the subpopulations. The parameters are important with regard to both the *shifting balance* and *punctuated equilibria* theories. Note that as the connectivity of the topology increases, i.e. tends toward a completely connected graph, and the frequency of interaction increases, i.e. the isolated evolution time for each $\mathcal{P}_i$ is shortened, the island model approximates more closely a single, large, freely intermixing population (see section C6.3.5). It is held generally that such large populations quickly reach stable gene frequencies, and thus, cease 'progress'. Eldredge and Gould (1972) termed this *stasis*, while the GA community generally refers to it as *premature convergence*.

At the other extreme, as the connectivity of the topology decreases, i.e. tends toward an edgeless graph, and the frequency of interaction decreases, i.e. each $\mathcal{P}_i$ has extended isolated evolution, the island model approximates more closely several independent trials of a sequential GA with a small population. We contend that such small populations 'exploit' strongly the area of local optima, but only those local optima extremely 'close' to the original population. Thus, intermediate degrees of connectivity and frequency of interaction provide the dynamics sufficient to allow both exploitation and exploration.

For our discussion here, the periods of isolated evolution will be called *epochs*, with migration occurring at the end of each epoch (except the last). The length of the epochs determines the frequency of interaction. Often the epoch length is specified by a number G_i of generations that $\mathcal{P}_i$ will evolve in isolation. However, a formulation more faithful to the theories of natural evolution would be to allow each subpopulation process to reach stasis, i.e. reach equilibrium or convergence, on each epoch (see section C6.3.5). From an implementation point of view with a subpopulation assigned to each processing element, this latter formulation allows the workload to become unbalanced and as such may be seen as an inefficient use of the parallel hardware if the processing elements having quickly converging subpopulations are forced to sit idle. The more troublesome problem is in measuring effectively the degree of stasis. 'Inefficiency' might occur when reasonably frequent, yet consistently marginal 'progress' is being made. Then not only might other processing elements be idle, but also the accumulated progress might not be worth the computation spent. In one of the experiments of the next section, we will present a system that incorporates an epoch-termination criterion. This system yields high-quality results more consistently, while being implemented in an overall parallel computing environment that utilizes the 'idle' processing elements.

The overall structure of the island model process comprises E major iterations called epochs. During an epoch each subpopulation process independently executes a sequential evolutionary algorithm for G_i generations. After each epoch there is a communication phase during which

individuals migrate between neighboring subpopulations. This structure is summarized in the following pseudocode:

```
Island_Model(E,N,μ)
{
  Concurrently for each of the i ← 1 to N subpopulations Initialize(P_i, μ);
  For epoch ← 1 to E do
      Concurrently for each of the i ← 1 to N subpopulations do
                  Sequential_EA(P_i,G_i);
      od;
      For i ← 1 to N do
                  For each neighbor j of i
                        Migration(P_i, P_j);
                  Assimilate(P_i);
      od
  od
  problem solution = best individual of all subpopulations;
}
```

Note that we specified **Sequential_EA** because the general framework can be applied to other evolutionary algorithms, e.g. evolution strategies (ES) (Lohmann 1990, Rudolph 1990).

After each phase of migration each subpopulation must assimilate the migrants. This assimilation step is dependent on the details of the migration process. For instance, in the implemented island model presented in the next section, if individual p_k is selected for emigration from $\mathcal{P}_i$ to $\mathcal{P}_j$ then p_k is deleted from $\mathcal{P}_i$ and added to $\mathcal{P}_j$. (The individual p_k itself migrates.) Also, the migration magnitudes, S_{ij}, are symmetric. Thus, the size of each subpopulation remains the same after migration and the assimilation is simply a fitness recalculation.

In other island models (Cohoon *et al* 1991a), if p_k is selected for emigration from $\mathcal{P}_i$ to $\mathcal{P}_j$ then p_k is added to $\mathcal{P}_j$ without being removed from $\mathcal{P}_i$. (A copy of individual p_k migrates.) This migration causes the subpopulation size to increase, so (under an assumption of a constant-size-subpopulation GA) the assimilation must include a reduction operation.

Still other parallel GAs (Mühlenbein *et al* 1987, Mühlenbein 1989, Gorges-Schleuter 1990, 1992, Tamaki 1992) implement overlapping subpopulations, i.e. the *diffusion model*. For such systems migration is not really an issue; rather its important effect is attained through the selection process. However, parallel GAs with overlapping subpopulations are best suited to medium-grained parallel architectures, and are discussed in Section B1.2.5.

C6.4

B1.2.5

An important aspect of both the *shifting balance* and *punctuated equilibria* theories is that the demes, or peripheral isolates, evolve in distinct environments. This aspect has two major facets. The first, and most obvious, facet is the restriction of the available breeding population, i.e. the isolated evolution of each subpopulation as we have already discussed. The second facet is the differing environmental attributes that determine the factors in natural selection. Wright has suggested that this facet provides 'ecological opportunity' (1982). For most GAs these factors, and the ways they interrelate to form the basis of natural selection, are encoded in the fitness function. Thus, to follow the fundamental analogy, the island model should have differing fitness functions at the various subpopulations. Of course, for GAs that are used for function optimization, the fitness function is almost always the objective function to be optimized (or some slight variation, e.g. an inversion to make an original minimization consistent with 'fitness'). We are not aware of any systems that have made use of this facet with truly distinct fitness functions among the subpopulations.

The island model presented in the next section (and many others) has a rudimentary form of this facet through *local* normalization of objective scores to yield fitness values. For example, an individual's fitness might be assigned to be its raw objective score divided by the current mean objective score across the given subpopulation. (This is the reason for the two fitness calculations in the pseudocode presented in section C6.3.4.2.) Such normalization does effect differing environments to the degree that the distributions of the individuals in the subpopulations differ.

For optimization problems that are multiobjective, there is usually a rather arbitrary linear weighting

of the various objective dimensions to yield a scalar objective score (see e.g. (C6.3.1)). This seems to provide a natural mechanism for having distinct objective functions, namely, distinct coefficient sets at each subpopulation. The difficulty of using this mechanism is that it clearly adds another level of evolution control parameters. Eldredge and Gould recognized this additional level when they discussed punctuated equilibria as a theory about the evolution of species, not a theory about the evolution of individuals within a species (Eldredge and Gould 1972). This is, indeed, an important facet of the island model; unfortunately, further exploration of its form and implications is beyond the scope of our discussion here.

C6.3.4 The island model genetic algorithm applied to a VLSI design problem

In order to illustrate the effectiveness of this parallel method and the effects of modifying important island model parameters, we present an island model applied to the routing problem in *VLSI circuit design*. In section C6.3.5, we then present the results from experiments in which the selected parameters were varied systematically and overall system performance evaluated. G3.5

C6.3.4.1 Problem formulation

The VLSI routing problem is defined as follows. Consider a rectangular routing region on a VLSI circuit with *pins* located on two parallel boundaries (*channel*) or four boundaries (*switchbox*). The pins that belong to the same *net* need to be connected subject to certain constraints and quality factors. The interconnections need to be made inside the boundaries of the routing region on a symbolic routing area consisting of horizontal *rows* and vertical *columns* (see figure C6.3.2).

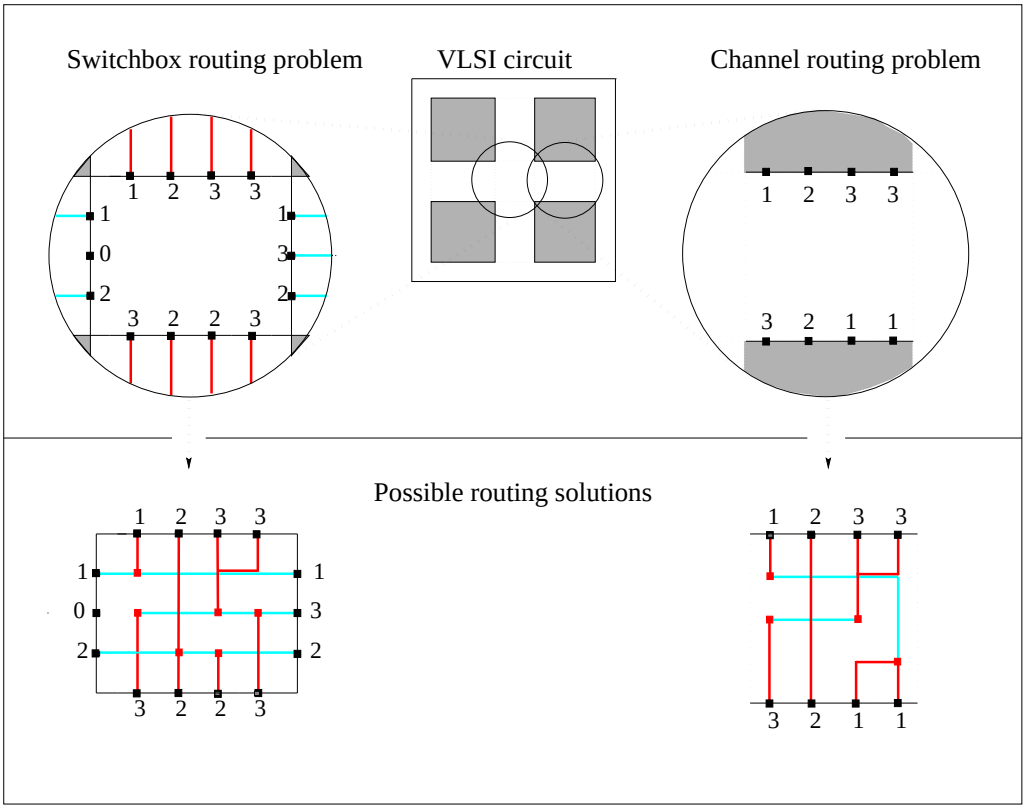


Figure C6.3.2. Example switchbox and channel routing problems ('magnified' in the circles) and possible routing solutions.

The routing quality of a particular solution involves (for the purposes of the following experiments) three factors: *netlength*—the shorter the length of the interconnections, the smaller the propagation delay, *number of vias*—the smaller the number of vias (connections between routing layers), the fewer electrical and fabrication problems occur, and *crosstalk*—in submicrometer regimes, crosstalk results mainly from

coupled capacitance between adjacent (parallel routed) interconnections, so the shorter these parallel-routed segments are, the less crosstalk occurs and the better the performance of the circuit. Thus, the optimization is to find a routing solution p_i for which $Obj(p_i)$ is minimal, with the objective function Obj specified by

$$Obj(p_i) = w_1 * l_{\text{nets}}(p_i) + w_2 * n_{\text{vias}}(p_i) + w_3 * l_{\text{par}}(p_i) \tag{C6.3.1}$$

where $l_{\text{nets}}(p_i)$ is the total length of the nets of p_i, $n_{\text{vias}}(p_i)$ is the number of vias of p_i, $l_{\text{par}}(p_i)$ is the total length of adjacent, parallel-routed net segments of p_i (crosstalk segments), and w_1, w_2, and w_3 are weight factors.

For VLSI designers it is important to have the weight factors, i.e. w_1, w_2, and w_3, to enable the designer to easily adjust routing quality characteristics: the netlength, the number of vias, and the tolerance of crosstalk, respectively, to the requirements of a given VLSI technology.

C6.3.4.2 The evolutionary process

As indicated in the pseudocode in section C6.3.3, each subpopulation process executes a sequential evolutionary algorithm. In this application we incorporated a genetic algorithm specified by the following pseudocode:

```
Sequential_GA(P_i, G_i)
{
  For generation ← 1 to G_i do
      P_new ← ∅;
      For offspring ← 1 to Max_offspring_i do
          p_α ← Selection(P_i);
          p_β ← Selection(P_i);
          P_new = P_new ∪ Crossover(p_α, p_β);
      od
      Fitness_calculation(P_i ∪ P_new);
      P_i ← Reduction(P_i ∪ P_new);
      Mutation(P_i);
      Fitness_calculation(P_i);
  od
}
```

Here $\mathcal{P}_i$ is the initial subpopulation that already includes any assimilated migrants from the last migration phase. In addition, the GA indicated above requires the following problem-domain-specific operators. For these operators, each individual is a complete routing solution, p_i.

Initialization. A random routing strategy (Lienig and Thulasiraman 1994) is used to create the initial subpopulations consisting of 'nonoptimized' routing solutions. These initial routing solutions are guaranteed to be feasible solutions, i.e. all necessary connections exist, but no refinement is performed on them. Thus, we consider them to be random solutions that are distributed throughout the search space.

Fitness calculation. The higher-quality solutions have smaller objective function values. So, to get a fitness value suitable for maximizing, a raw fitness function is calculated as the inverse of the objective function (see equation (C6.3.1)), $F'(p_i) = 1/Obj(p_i)$, then the final fitness $F(p_i)$ of each individual p_i is determined from $F'(p_i)$ by linear scaling (Goldberg 1989) *local* to the specific subpopulation.

Selection. The selection strategy, which is responsible for choosing mates for the crossover procedure, is *stochastic sampling with replacement* (Goldberg 1989); that is, individuals are selected with probabilities proportional to their fitness values. C2.2

Crossover. Two individuals are combined to create a single offspring. The crossover operator gives high-quality routing components of the parents an increased probability of being transferred intact to their offspring (low disruption). The operator is analogous to *one-point crossover*, with a randomly positioned C3.3

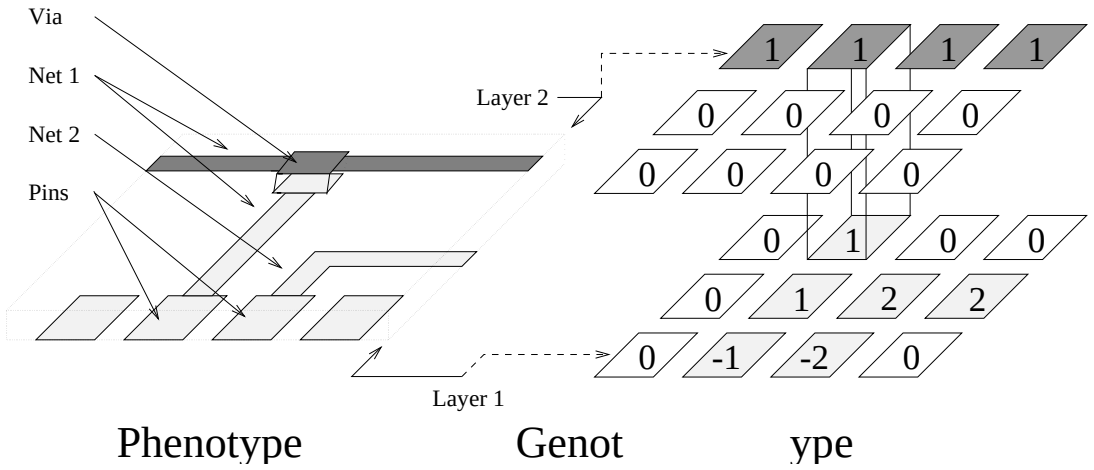


Figure C6.3.3. Representing a routing solution (the phenotype) as a three-dimensional chromosome (the genotype).

line (a horizontal or vertical crossline) that divides the routing area into two sections, playing the role of the crosspoint. For example, net segments *exclusively* on the upper side of a horizontal crossline are inherited from the first parent, while segments *exclusively* on the lower side of the crossline are inherited from the second parent. Net segments intersecting the crossline are newly created for the offspring by means of a random routing strategy (the same strategy as used in *Initialization*). The full details of this operator (Lienig and Thulasiraman 1994) are beyond the scope of this discussion, but figure C6.3.3 provides a general idea of how each individual routing solution is represented. In the genotype, the routing surface is represented by an 'occupancy' model, that is, each unit of surface area is represented by a node (a circle in figure C6.3.3). Nodes are connected if their corresponding surface areas are adjacent, either within a layer or across layers. The value at a node indicates which net is routed through that surface area, with a negative value indicating a pin (thus fixed assignment) position and zero indicating that the area is unused.

Mutation. The mutation operator performs random modifications on an individual, i.e. changes the routing solution randomly. The purpose is to overcome local optima and to exploit new regions of the search space (Lienig and Thulasiraman 1994).

Reduction. The reduction strategy combines the current subpopulation with the newly created set of offspring, it then simply chooses the fittest individuals from the combined set to be the subpopulation in the next generation, thus keeping the subpopulation size constant.

C6.3.4.3 Parallel structure

The island model used for this application has nine subpopulations ($N = 9$) connected by a torus topology. Thus, each subpopulation has exactly four *neighbors*. The total population ($M = 450$) is evenly partitioned ($\mu = 50$). (The listed numbers will be changed in the course of the experiments discussed in the following.) The parallel algorithm has been implemented on a network of SPARC workstations (SunOS and Solaris systems). The parallel computation environment is provided by the Mentat system, an object-oriented parallel processing system (Grimshaw 1993, Mentat 1996). The program, written in C++ and Fortran, comprises approximately 10 000 lines of source code. The cost factors in (C6.3.1) are set to $w_1 = 1.0$, $w_2 = 2.0$, and $w_3 = 0.01$. The experimental results were achieved with the machines running their normal daily loads in addition to this application.

C6.3.4.4 Comparison to other routing algorithms

Any experiment involving a 'real' application of an evolutionary algorithm should begin with a comparison to solution techniques that have already been acknowledged as effective by that application's community. Here we will simply state the results of the comparison because the detailed numbers will only be

meaningful to the VLSI routing community and have appeared elsewhere (Lienig 1996). First, 11 benchmark problem instances were selected for channel and switchbox routing problems, for example, *Burstein's difficult channel* and *Joo6_16* for channels and *Joo6_17* and *Burstein's difficult switchbox* for switchboxes. These benchmarks were selected because published results were available for various routing algorithms, namely, Yoshimura and Kuh (1982), WEAVER (Joobbani 1986), BEAVER (Cohoon and Heck 1988), PACKER (Gerez and Herrmann 1989), SILK (Lin *et al* 1989), Monreale (Geraci *et al* 1991), SAR (Acan and Ünver 1989), and PARALLEX (Cho *et al* 1994). Note that most of these systems implement deterministic routers.

The island model was run 50 times per benchmark (with varying parameters) and the best-seen solution for each benchmark recorded. A comparison of those best-seen solutions to the previously published best-known solutions indicates that the island model solutions are qualitatively equal to or better than the best-known solutions from channel and switchbox routers for these benchmarks.

Of course, due to the stochastic nature of a GA, the best-seen results of the island model were not achieved in every run. (All executions were based on arbitrary initializations of the random number generator.) Above we refer to the *best-seen* solutions over all the runs for each benchmark; however, we would like to note that, in fact, in at least 50% of the individual island model runs solutions equal to these best-seen results were obtained. We judge this to be very consistent behavior for a GA.

C6.3.5 The influence of island model parameters on evolution

Several experiments have been performed to illustrate the specific effects of important island model parameters in order to guide further applications of coarse-grained parallel GAs. The specific parameters varied in the experiments are the magnitude of migration, the frequency of migration, the epoch termination criterion, the migrant selection strategy, and the number of subpopulations and their sizes.

Five benchmark problem instances, namely, *Burstein's difficult channel*, *Joo6_13* and *Joo6_16* for channels and *Joo6_17*, and *pedagogical switchbox* for switchboxes, were chosen for these experiments. Comparisons were made between various parameter settings for the island model. In addition, runs were made with a sequential genetic algorithm (SGA) and a strictly isolated island model, i.e. no migration. The SGA executed the same algorithm as the subpopulation processes, but with a population size equal to the sum over all subpopulation sizes, i.e. $N * \mu$.

In the experiments, the SGA was set to perform the same number of recombinations per generation as the island model does over all subpopulations, namely, (number of subpopulations) $\times$ (offspring per subpopulation). The SGA and island model configurations were run for the same total number of generations. Thus, we ensure a fair method (with regard to the total number of solutions generated) to compare our parallel approach with an SGA.

The fundamental baseline was a derived measure based on the best-known objective measure for each problem instance. Remember that for the objective function, smaller values indicate higher-quality solutions. The derived measure is referred to as Δ and is calculated as indicated in the following. Let R_{bk} be the objective measure of the best-known solution and let R_{SGA} be the best-seen result on a particular run of SGA. Then,

$$\delta_{\mathrm{SGA}} = \frac{R_{\mathrm{SGA}} - R_{\mathrm{bk}}}{R_{\mathrm{bk}}} \tag{C6.3.2}$$

is a relative (to the best-known) difference for a single run of SGA. The δ_{SGA} were averaged over five runs for each benchmark and over the five benchmarks to yield Δ_{SGA}.

In figures C6.3.4 and C6.3.6–C6.3.8, this Δ_{SGA} is shown as a 100% bar in the leftmost position in the plot. Similar Δ values were obtained for the various island model configurations and are shown as percentages of Δ_{SGA}. Thus, if a particular island model configuration is shown with a 70% bar, then the average relative difference for that configuration is 30% better than SGA's average relative difference from the best-known result.

This derived measure was used in order to combine comparisons across problem instances with disparate objective function value ranges. In addition, the measure establishes a baseline through both best-known and SGA results. We remind the reader that for each benchmark problem this island model system evolved a solution equal to or better than any previously published system.

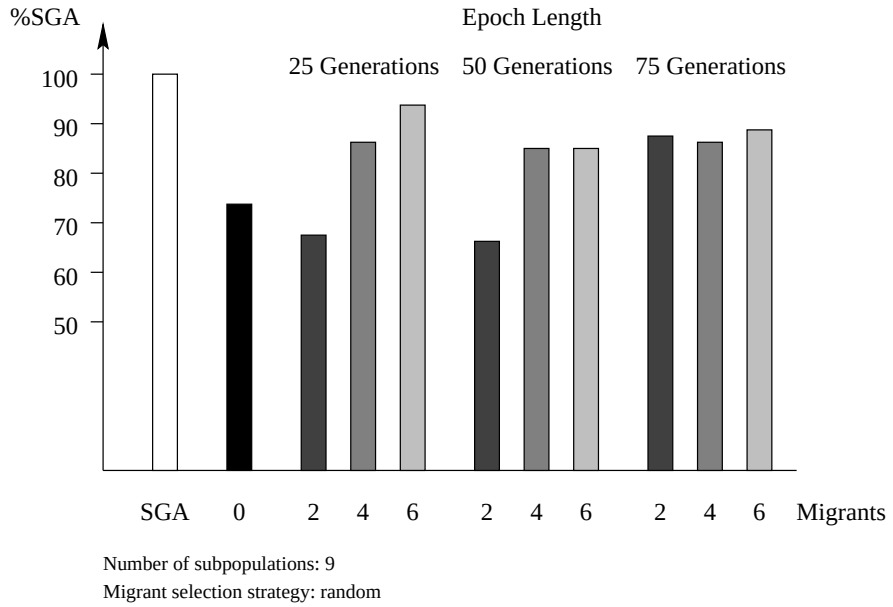


Figure C6.3.4. A comparison of results on the benchmark suite with different numbers of migrants and epoch lengths. Each bar is a Δ value for a different configuration. A Δ value is an average relative difference from the best-known solution as normalized to the Δ_{SGA} value, so the SGA bar is always 100%. Thus, the lower the bar, the better the average result of the particular configuration.

C6.3.5.1 Number of migrants and epoch lengths

We investigated the influence of different epoch lengths (number of generations between migration) for different numbers of migrants (number of individuals sent to each of the four neighbors). The migrants were chosen randomly, with each migrant allowed to be sent only once. Figure C6.3.4 shows that the sequential approach was outperformed by all parallel variations (when averaged over all considered benchmarks). Note that the set of parallel configurations included the version with no migration, i.e. the strictly isolated island model (shown in figure C6.3.4 as '0 migrants'). Thus, the splitting of the total population size into independent subpopulations has already increased the probability that at least one of these subpopulations will evolve toward a better result (given at least a 'critical mass' at each subpopulation, as we discuss in section C6.3.5.4).

Figure C6.3.4 also shows that a *limited* migration between the subpopulation further enhance the advantage of a parallel genetic algorithm. Two migrants to each neighbor with an epoch length of 50 generations are seen to be the best parameters when averaged over all problem instances. On the one hand, more migrants or too short epoch lengths are counterproductive to the idea of disjointly and parallel evolving subpopulations. The resulting intermixing diminishes the genetic diversity between the subpopulations by 'pulling' them all into the same part of the search space, thereby approaching the behavior of a single-population genetic algorithm. On the other hand, insufficient migration (an epoch length of 75 generations) simulates the isolated parallel approach (zero migrants)—the genetic richness of the neighboring subpopulations does not have enough chance to spread out.

Figure C6.3.5 shows this behavior in the context of individual subpopulations; that is, it presents the convergence behavior of the best individuals in each of the parallel evolving subpopulations on a specific problem instance (channel Joo6_13). It clearly indicates the importance of migration to avoid premature stagnation by infusing new genetic material into a stagnating subpopulation. The 'stabilizing' effect of migration is also evident in the reduced variation among the best objective values gained in five independent runs, as shown in the right-hand plot of figure C6.3.5.

C6.3.5.2 Variable epoch lengths

The theory of punctuated equilibria is based on two main ideas: (i) an isolated subpopulation in a constant environment will stabilize over time with little motivation for further development and (ii) continued evolution can be obtained by introducing new individuals from other, also stagnating subpopulations.

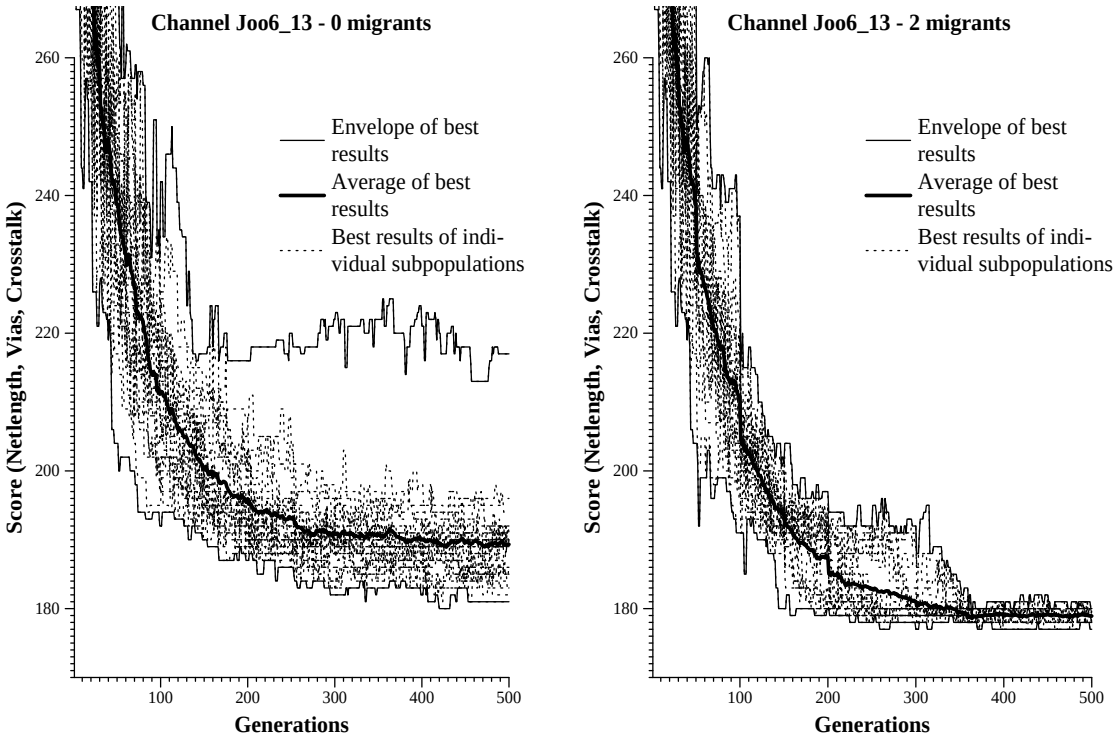


Figure C6.3.5. A comparison of the convergence of the best solutions in the individual, parallel evolving subpopulations. Plotted are five runs with nine subpopulations, i.e. 45 runs, in isolation (left) and with two migrants (right). (Note that the envelope for the plot on the left looks unusual due to an outlier subpopulation.)

However, all known computation models that are based on this theory use a fixed number of generations between migration. Thus, they do not exactly duplicate the model that migration occurs only *after* a stage of equilibrium has been reached within a subpopulation.

The algorithm was modified to investigate the importance of this characteristic. Rather than having a fixed number of generations between migrations, a stop criterion was introduced that took effect when stagnation in the convergence behavior within a subpopulation had been reached. After some experimentation with different models, we defined a suitable stop criterion to be 25 generations with no improvement in the best individual within a subpopulation.

To ensure a fair comparison, we kept the overall number of generations the same as in all other experiments. This led to varying numbers of epochs between the parallel evolving subpopulations (due to different epoch lengths) and resulted in longer overall completion time.

The results achieved with this variable epoch length are shown in figure C6.3.6. The results suggest that a slight improvement compared with a fixed epoch length can be achieved by this method. However, it is important to note that this comparison is made with a fixed epoch length that has been shown to be the most suitable after numerous experiments (see figure C6.3.4). Thus, the important attributes to notice are that the variable-epoch-length configuration frees the user from finding a suitable epoch length and that it gave more consistent results over the various migration settings.

C6.3.5.3 Different migrant selection strategies

The influence of the quality of the migrants on the routing results was investigated using three migrant selection strategies: 'random' (migrants were chosen randomly with uniform distribution among the entire subpopulation), 'top 50%' (migrants were chosen randomly among the individuals with a fitness above the median fitness of the subpopulation), and 'best' (only the best individuals of the subpopulation migrated). The migrants were sent in a random order to the four neighbors.

As figure C6.3.7 indicates, we cannot find any improvement in the obtained results by using migrants with better quality. On the contrary, selecting better (or the best) individuals to migrate leads to a faster

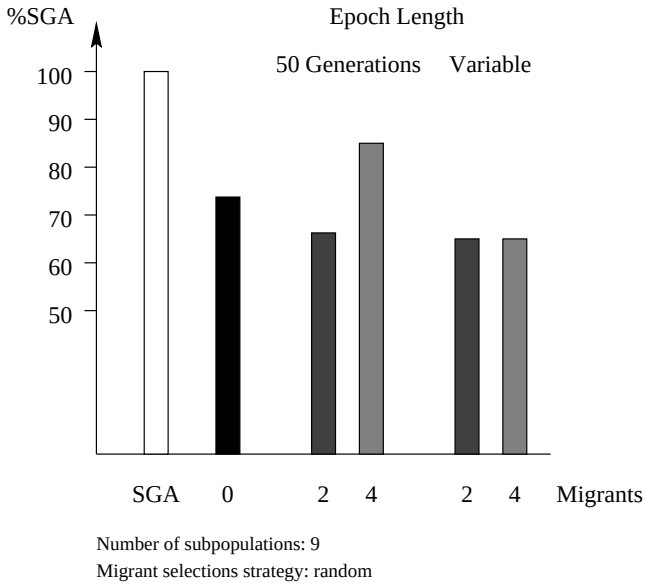


Figure C6.3.6. A comparison of results on the benchmark suite with fixed and variable epoch lengths. Variable-length epochs were terminated after 25 generations of no improvement of the best individual within the subpopulation. Each bar is a Δ value.

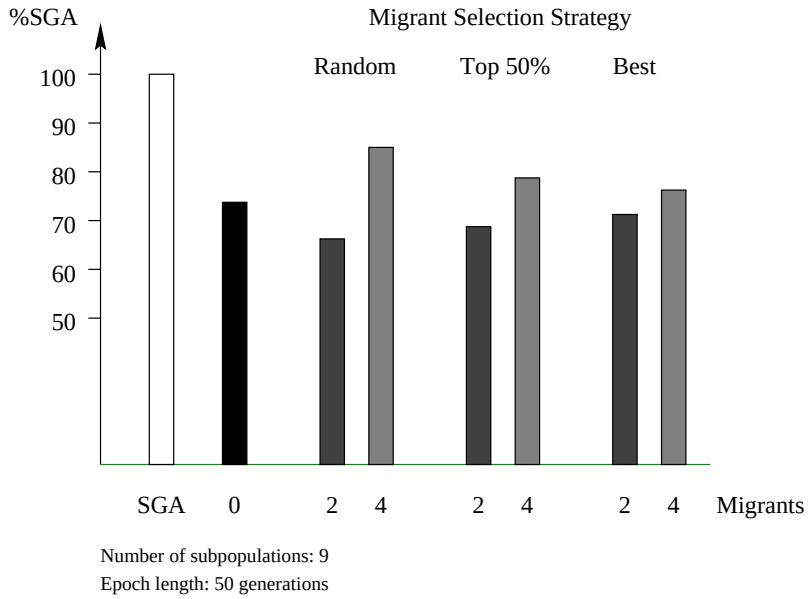


Figure C6.3.7. A comparison of results on the benchmark suite with different migrant selection strategies. Each bar is a Δ value.

convergence—the final results are not as good as those achieved with a less elitist selection strategy. According to our observations, this is due to the dominance of the migrants having their presently superior genetic material reach all the subpopulations, thus leading the subpopulation searches into the same part of the search space concurrently.

C6.3.5.4 Different numbers of subpopulations

To compare the influence of the number of subpopulations, the sizes of the subpopulations were kept constant and the number of subpopulations increased from $N = 9$ to $N = 16$ and $N = 25$ (still connected in a torus). Accordingly, the population size and the number of recombinations of the SGA were increased to maintain a fair comparison. The resulting plots for SGA against 16 and 25 subpopulations (with $\mu = 50$)

are qualitatively similar to the SGA against nine subpopulations comparison of figure C6.3.4. For problems of this difficulty one would expect the SGA with slightly larger populations to do better than the small-population SGAs. That expectation was indeed born out in these experiments. The important observation is that the island model performance also increased; thus the relative performance advantage of the island model was maintained.

In an interesting variation on this experiment, the total population size, M, was held constant while increasing the number of subpopulations (and so reducing the subpopulation sizes). Holding M near 450 and increasing N to 16 yielded $\mu = 28$, while increasing N to 25 yielded $\mu = 18$.

The results presented in figure C6.3.8 show that subpopulation size is an important factor. The figure clearly indicates that (for this average measure) partitioning a population into subpopulations yielded (for $N = 9$) results better than SGA, then yielded progressively worse results as N was increased (and μ was decreased). We contend that this progression was due to the subpopulation size, μ, falling below 'critical mass' for the specific benchmark problem instances. Remember, the plotted values are aggregate measures. When we looked at the component values for each problem instance we found further evidence for our contention. The evidence was that for the simpler benchmarks the $N = 25$ island model still had extremely good performance, in fact, equaled the best-known solution repeatedly. For the other, more complex benchmarks, the $N = 25$ island model performed very poorly, thus dragging the average measure well below the SGA performance level. Thus, the advantage of more varied evolving subpopulations can be obtained by increasing N only if μ remains above 'critical mass'. This critical value for μ is dependent on the complexity of the problem instance being solved.

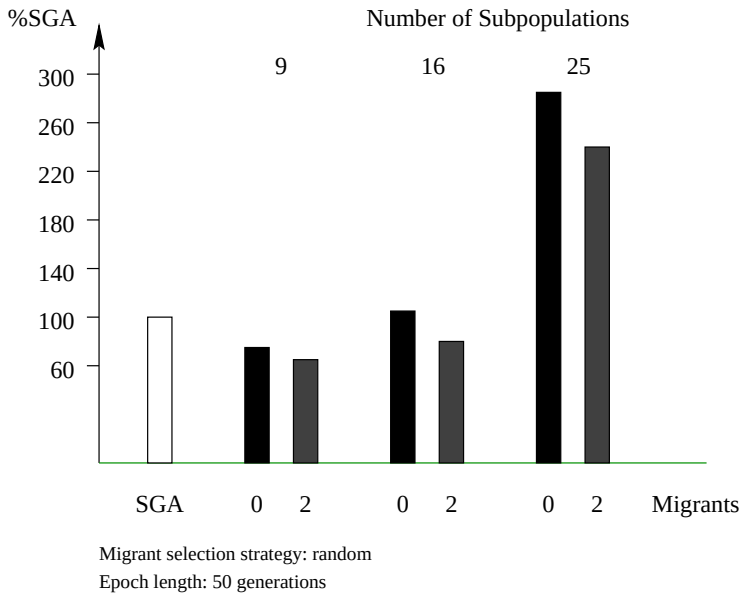


Figure C6.3.8. A comparison of results on the benchmark suite with different numbers of subpopulations. The size of the total population, M, is kept constant at 450 individuals. Since $M = N * \mu$, the increase in N, the number of subpopulations, requires a reduction in μ, the size of each subpopulation.

C6.3.6 Final remarks and conclusions

In general it is difficult to compare sequential and parallel algorithms, particularly for stochastic processes such as GAs. As mentioned at the beginning of our discussion, the comparison is often according to overall time to completion, i.e. wall clock time. We contend that the island model constitutes a different evolutionary algorithm, not just a faster implementation of the SGA, and one that yields qualitatively better results. Here we have argued for this contention from the point of view of biological evolution and in the context of a difficult VLSI design application.

Several of the experimental design decisions we made for the application experiments merit reiteration here. First, the application is an important problem with an extensive literature of heuristic systems that 'solve' the problem. Our derived baseline measure incorporated the best-known objective values from

this literature (see (C6.3.2)). For this particular VLSI design problem, most of the heuristic systems are deterministic: thus we have aggregate values for the various GAs versus single values for the deterministic systems. Our measure does not directly account for this but we have provided an indication of the variation associated with the set of runs for the island model.

Second, our comparisons to an SGA are based on 'best-seen' objective values, not central processing unit (CPU) time or time to completion. In order to make these comparisons fair, we have endeavored to hold the computational resources constant and make them consistent across the SGA and the various configurations of the island model. This was done by fixing the total number of recombinations, i.e. the number of applications of the crossover operator, which relates directly to the total number of objective function evaluations.

Using the number of recombinations, as opposed to CPU time, for example, allows us to ignore properly implementation details for subsidiary processes such as sorting or insertion into a sorted list (Garey and Johnson 1979). Note that a CPU time measure can 'cut both ways' between the serial and parallel versions. On the one hand, if a subsidiary process has a small initial constant but poor performance as the data structure size increases, then the island model has an advantage simply through the partition of the population. On the other hand, if a subsidiary process is increasingly efficient for larger data structures but has a large initial constant, then the SGA has the advantage, again, simply through the partition of the population. These are examples of the subtle ways by which time-based comparisons confound primary search effort with irrelevant implementation details.

Third, our comparisons have ignored the cost of communication. This is generally appropriate for island models because the isolated computation time for each subpopulation is extremely large relative to the communication time for migration (any course-grained parallelization should have this attribute). For medium- and fine-grained parallel models communication is much more of a real concern. Ignoring communication time is also reflective of our interest in the evolutionary behavior of the models, not the raw speed of a particular implementation.

With all GAs, the evolutionary behavior depends heavily on the interplay between the problem complexity and population size. For SGA against island model comparisons this is particularly problematic because the island model has two population sizes: the total population size, M, and the subpopulation size, μ. Which is the proper one to consider relative to the SGA population size? For our experiments, we have used the total population size. We believe this makes the versions more conformable and is most consistent with number of recombinations as the computation measure.

Further, for comparing stochastic processes, such as GAs, the total number of trials is important, particularly when the evaluation measure is based on best-seen results. The attentive reader will have noticed that the strictly isolated island model (no migration) often does better than the SGA. This might seem curious, since a single run of the isolated island model is just a set of N separate SGAs and ones with smaller population sizes. As long as the subpopulation size, μ, is above what we are calling the 'critical mass' level, the isolated island model has a statistical advantage over the single SGA (under our evaluation measure).

The evaluation measure gives this advantage because the best seen is taken over each run. Thus, each run of the isolated island model has N 'samples' to determine its best seen, while each SGA run has only one 'sample'. (Shonkwiler (1993) calls these *IIP parallel* GAs and gives an analysis of 'hitting time' expectation.) We consider the 'samples' in this case to be evolutionary trajectories through the solution space. Now, note that in almost all cases allowing migration provides the island model with the means to derive even better results.

This application of the island model to detailed routing problems in VLSI circuit design has shown that a parallel GA based on the theory of *punctuated equilibria* outperforms an SGA. Furthermore, the results are qualitatively equal to or better than the best-known results from published channel and switchbox routers.

In investigating the parameters of the island model, the following conclusions have been reached:

- The island model consistently performs better than the SGA, given a consistent amount of computation.
- The size of a subpopulation, the total amount of immigration, i.e. the number of connected subpopulations multiplied by the number of migrants per neighbor, the epoch length and the complexity of the problem instance are interrelated quantities. The problem instance complexity determines a minimum population size for a viable evolutionary trajectory. The total amount of immigration must not be disruptive (our experiments indicate that more than 25% of the subpopulation size is disruptive), and the epoch length must be long enough to allow exploitation of

the infused genetic material. Within these constraints the island model will perform better with more subpopulations, even while holding the total population size and the total number of recombinations, i.e. amount of computation, constant.

- Variable epoch lengths determined via equilibrium measures within subpopulations achieve overall results slightly better than those obtained with (near-) optimized fixed epoch lengths. Though an equilibrium measure must be chosen, allowing variable epoch lengths frees the user from having to select this parameter value.
- Quality constraints on the migrants do not improve the overall behavior of the algorithm: on the contrary, quality requirements on the selection of the migrants increases the occurrence of premature stagnation.
- Given a sufficient number of individuals per subpopulation, the larger the number of parallel evolving subpopulations, the better the routing results. The complexity of the problem and the minimal subpopulation size have a direct correlation that must be taken into account when dividing a population into subpopulations.

Finally, we would like to return to an issue that we mentioned at the very beginning of our discussions: namely, the island model formulation of the GA is not simply a hardware accelerator of the single-population GA. The island model does map naturally to distributed-memory message-passing multiprocessors, so it is amenable to the speedup in time to completion that such parallel architectures can provide. However, the formulation can improve the quality of solutions obtained even via sequential simulations of the island model. As supported by the *shifting balance* and *punctuated equilibria* theories, the emergent properties of the computation derive from the *concurrent* evolutionary trajectories of the subpopulations *interacting* through limited migration.

References

Acan A and Ünver Z 1992 Switchbox routing by simulated annealing: SAR *Proc. IEEE Int. Symp. on Circuits and Systems* vol 4, pp 1985–8

Adamidis P 1994 *Review of Parallel Genetic Algorithms* Technical Report, Department of Electrical and Computer Engineering, Aristotle University, Thessaloniki

Bailey J 1992 First we reshape our computers, then our computers reshape us: the broader intellectual impact of parallelism *Daedalus* pp 67–86

Barnes G H, Brown R M, Kato M, Kuck D J, Slotnick D L and Stokes R A 1968 The ILLIAC IV computer *IEEE Trans. Computer* **C-17** 746–57

Cho T W, Sam S, Pyo J and Heath R 1994 PARALLEX: a parallel approach to switchbox routing *IEEE Trans. Computer-Aided Design* **CAD-13** 684–93

Cohoon J P and Heck P L 1988 BEAVER: a computational-geometry-based tool for switchbox routing *IEEE Trans. Computer-Aided Design* **CAD-7** 684–97

Cohoon J P, Hegde S U, Martin W N and Richards D S 1987 Punctuated equilibria: a parallel genetic algorithm *Proc. 2nd Int. Conf on Genetic Algorithms (Pittsburgh, PA, 1987)* ed J J Grefenstette (Hillsdale, NJ: Erlbaum) pp 148–54

——1991a Distributed genetic algorithms for the floorplan design problem *IEEE Trans. Computer-Aided Design* **CAD-10** 483–92

Cohoon J P, Martin W N and Richards D S 1991b Genetic algorithms and punctuated equilibria in VLSI *Parallel Problem Solving from Nature (Lecture Notes in Computer Science 496)* ed H P Schwefel and R Männer (Berlin: Springer) pp 134–44

Eldredge N 1989 *Macro-evolutionary Dynamics: Species, Niches, and Adaptive Peaks* (New York: McGraw-Hill)

Eldredge N and Gould S J 1972 Punctuated equilibria: an alternative to phyletic gradualism *Models of Paleobiology* ed T J M Schopf (San Francisco, CA: Freeman, Cooper) pp 82–115

Forrest S (ed) 1991 *Emergent Computation* (Cambridge, MA: MIT Press)

Fung L W 1976 *MPPC: a Massively Parallel Processing Computer* Goddard Space Flight Center Section Report

Garey M R and Johnson D S 1979 *Computers and Intractability: a Guide to the Theory of NP-completeness* (San Francisco, CA: Freeman)

Geraci M, Orlando P, Sorbello F and Vasallo G 1991 A genetic algorithm for the routing of VLSI circuits *Proc. Euro ASIC '91* pp 218–23

Gerez S H and Herrmann O E 1989 Switchbox routing by stepwise reshaping *IEEE Trans. Computer-Aided Design* **CAD-8** 1350–61

Goldberg D E 1989 *Genetic Algorithms in Search, Optimization, and Machine Learning* (Reading, MA: Addison-Wesley)

Gordon V S, Whitley D and Böhn A 1992 Dataflow parallelism in genetic algorithms *Parallel Problem Solving from Nature, 2 (Brussels 1992)* ed R Männer and B Manderick (Amsterdam: Elsevier Science) pp 533–42

Gorges-Schleuter M 1990 Explicit parallelism of genetic algorithms through population structures *Parallel Problem Solving from Nature (Lecture Notes in Computer Science 496)* ed H-P Schwefel and R Männer (Berlin: Springer) pp 150–9

Gorges-Schleuter M 1992 Comparison of local mating strategies *Parallel Problem Solving from Nature, 2 (Brussels 1992)* ed R Männer and B Manderick (Amsterdam: Elsevier Science) pp 553–62

Grimshaw A S 1993 Easy-to-use object-oriented parallel programming with Mentat *IEEE Computer* **26** 39–51

Grosso P 1985 *Computer Simulation of Genetic Adaptation: Parallel Subcomponent Interaction in a Multilocus Model* PhD Thesis, Computer and Communication Sciences Department, University of Michigan

Holland J H 1975 *Adaptation in Natural and Artificial Systems* (Ann Arbor, MI: University of Michigan Press)

Huxley J 1942 *Evolution: the Modern Synthesis* (New York: Harper)

Joobbani R 1986 *An Artificial Intelligence Approach to VLSI Routing* (Boston, MA: Kluwer)

Lienig J 1996 A parallel genetic algorithm for two detailed routing problems *IEEE Int. Symp. on Circuits and Systems (Atlanta, GA, 1996)* pp 508–11

Lienig J and Thulasiraman K 1994 A genetic algorithm for channel routing in VLSI circuits *Evolutionary Comput.* **1** 293–311

Lin Y-L, Hsu Y-C and Tsai F-S 1989 SILK: a simulated evolution router *IEEE Trans. Computer-Aided Design* **CAD-8** 1108–14

Lohmann R 1990 Application of evolution strategies in parallel populations *Parallel Problem Solving from Nature (Lecture Notes in Computer Science 496)* ed H P Schwefel and R Männer (Berlin: Springer) pp 198–208

Mentat 1996 homepage http://www.cs.virginia.edu/~mentat/

Mühlenbein H 1989 Parallel genetic algorithms, populations genetics and combinatorial optimization *Proc. 3rd Int. Conf on Genetic Algorithms (Fairfax, VA, 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 416–21

Mühlenbein H, Gorges-Schleuter M and Krämer O 1987 New solutions to the mapping problem of parallel systems—the evolution approach *Parallel Comput.* **6** 269–79

Pettey C B, Leuze M R and Grefenstette J J 1987 A parallel genetic algorithm *Proc. 2nd Int. Conf. on Genetic Algorithms (Pittsburgh, PA, 1987)* ed J J Grefenstette (Hillsdale, NJ: Erlbaum) pp 155–61

Radcliffe N J 1991 Forma analysis and random respectful recombination *Proc. 4th Int. Conf on Genetic Algorithms (San Diego, CA, 1991)* ed R K Belew and L B Booker (San Mateo, CA: Morgan Kaufmann) pp 222–9

Rudolph G 1990 Global optimization by means of distributed evolution strategies *Parallel Problem Solving from Nature (Lecture Notes in Computer Science 496)* ed H P Schwefel and R Männer (Berlin: Springer) pp 209–13

Seitz C L 1985 The cosmic cube *Commun. ACM* **28** 22–33

Shonkwiler R 1993 Parallel genetic algorithms *5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 199–205

Slotnick D, Borck W and McReynolds R 1962 The SOLOMON computer *Proc. Fall Joint Computer Conf.* (AFIPS) pp 97–107

Tamaki H and Nishikawa Y 1992 A parallel genetic algorithm based on a neighborhood model and its application to the jobshop scheduling *Parallel Problem Solving from Nature, 2 (Brussels 1992)* ed R Männer and B Manderick (Amsterdam: Elsevier Science) pp 573–82

Tanese R 1987 Parallel genetic algorithms for a hypercube *Proc. 2nd Int. Conf on Genetic Algorithms (Pittsburgh, PA, 1987)* ed J J Grefenstette (Hillsdale, NJ: Erlbaum) pp 177–83

Wright S 1932 The roles of mutation, inbreeding, crossbreeding and selection in evolution *Proc. 6th Int. Congr. Genetics* vol 1, pp 356–66

——1964 Stochastic processes in evolution *Stochastic Models in Medicine and Biology* ed J Gurland (Madison, WI: University of Wisconsin Press) pp 199–241

——1982 Character change, speciation, and the higher taxa *Evolution* **36** 427–43

Wulf W A and Bell C G 1972 C.mmp—a multi-mini-processor *Proc. Fall Joint Conf* (AFIPS) pp 765–77

Yoshimura T and Kuh E S 1982 Efficient algorithms for channel routing *IEEE Trans. Computer-Aided Design* **CAD-1** pp 25–35

C6.4 Diffusion (cellular) models

Chrisila C Pettey

Abstract

When considering the parallel nature of any evolutionary algorithm, perhaps the first thing that comes to mind is the fact that each individual in the population is a separate trial in the search space. In that sense, one generation is a parallel evaluation of μ points in the search space. Similarly, when considering evolution from a population genetics point of view, it appears that neither selection nor mutation and definitely not recombination occur in a global population sense. Rather, these evolutionary operations appear to occur in demes—'locally interbreeding groups of organisms' (Hartl 1980). In other words, it seems that 'the evolution process is driven by the individuals themselves' (Mühlenbein 1989), and that the genetic makeup of the individuals will spread throughout the global population in a manner similar to the diffusion process (Gorges-Schleuter 1989). This idea of the individual being the parallel part of the evolution process is the basic concept behind the diffusion model. The purpose of this section is to describe the diffusion model in detail. To that end it begins with a description of the diffusion model, continues with a discussion of implementation techniques, and concludes with a brief discussion of some theoretical work in the area of diffusion models.

C6.4.1 A formal description of the diffusion model

Since a generation is the parallel evolution of μ individuals, the diffusion model is implemented by placing one individual per processor. Thus, μ is determined by and is equal to the number of processing nodes in the parallel environment. Because of this limitation, the population size remains constant at all times. Furthermore, since the idea of a generation involves mating, the diffusion model generally contains some form of recombination. Given these two limitations of fixed population size and recombination, the diffusion model is almost always a *genetic algorithm* (GA) model. B1.2

In the diffusion model, each process is responsible for evolving its individual. In keeping with the deme theory, while selection and recombination could be performed globally, they generally are performed within a local neighborhood. Thus, the pseudocode for a single process in this model might appear as follows:

```
Process (i):
    1. t ← 0;
    2. initialize individual i;
    3. evaluate individual i;
    4. while (t ≤ t_max) do
    5.         individual i ← select(neighborhood(i))
    6.         choose parent 1 from neighborhood
    7.         choose parent 2 from neighborhood
    8.         individual i ← recombine(parent1, parent2)
    9.         individual i ← mutate(individual i)
    10.        evaluate individual i;
    11.        t ← t + 1;
       od
```

It has been proved by Whitley (1993) that any evolutionary algorithm (EA) of this form is equivalent to a cellular automaton. Thus, the diffusion model could also be called the cellular model. As in all EA models, in order to implement this model, it will be necessary to determine what an individual will look like, how an individual will be initialized and evaluated, what will be the mutation rate, and what will be the stopping criteria. All of these topics are discussed in other sections. On the other hand, the key implementation issues that are unique to the diffusion model—how should selection be performed, how are the parents chosen, what is the size and shape of the neighborhood, and how is recombination performed—along with several techniques for implementing each of these issues are presented in the remainder of this section. It should be noted, that since evolution is viewed on an individual basis, throughout the rest of this section the diffusion model will be viewed from the perspective of a single process which is evolving a single individual.

C6.4.2 Diffusion model implementation techniques

When considering how to implement the diffusion model, it is tempting to perform a global selection and recombination because then the model would theoretically be equivalent to a sequential EA. This solution is made even more inviting by the fact that all massively parallel machines have a front-end processor or a host processor for handling global operations. The problem with this solution is that in a parallel machine environment global operations are much more time consuming than local operations. The problem is magnified in a distributed environment. To overcome the inefficiency of the global solution and to maintain the idea of the individual being the parallel part of the process, almost all implementations (for a counterexample see Farrell *et al* (1994)) perform selection and recombination in the local neighborhood (Collins and Jefferson 1991). Furthermore, since a process is only responsible for its individual, selection can be combined with the process of choosing the parents for recombination. In most implementations that perform selection and recombination in separate steps, the selection is a local implementation of a global technique such as proportional, ranking, or binary tournament selection (see for example Collins and Jefferson 1991, De Jong and Sarma 1995). The local versions of the global techniques are implemented just like the global techniques with the exception that the individual's population is just the neighborhood—not the global population. Since *selection* is discussed thoroughly in a previous chapter, it will not be C2 discussed here. In this section, techniques for choosing parents, neighborhood size and shape attributes, and recombination techniques are presented. However, since many techniques and choices may be based on the parallel environment, it is necessary to begin with a short description of typical parallel environments for the diffusion model.

C6.4.2.1 Parallel environments for diffusion model implementation

Michael Flynn (1966) coined the terms MIMD (multiple instruction–multiple data) and SIMD (single instruction–multiple data), which are used to characterize parallel processing paradigms. Both terms are typically used to characterize hardware paradigms, but, perhaps just as typically, the terms are also applied to the algorithms that are written to exploit the underlying hardware. A third term, SPMD (single program–multiple data), is a term that is applied to the situation where the algorithm is a SIMD algorithm, but the underlying hardware is MIMD.

In a MIMD environment, processors work independently and asynchronously. Coordination and communication among processes in the form of locks, barriers, shared variables, or message passing are handled by the programmer. Some examples of MIMD machines are hypercubes, Thinking Machines CM-5, and the Intel Paragon. Another MIMD environment that is becoming more common is clusters of workstations (or farms) running a parallel software system such as PVM, Linda, or Express. A MIMD program consists of two or more (usually different) processes. In this sense, it is the functions which are executed in parallel. Thus, this form of parallelism is usually called functional (or course-grained) parallelism. With the advent of the hypercube in the mid-1980s, the MIMD computing environment became accessible to more people. As a result, the *island (migration) model*, which is well suited to a C6.3 MIMD environment, was more popular than the diffusion model.

In the last decade, however, SIMD machines have become more readily available and more user friendly. Some typical SIMD machines are Thinking Machines CM-1 and CM-2, Active Memory Technology Ltd DAP, and MasPar MP1 and MP2. All SIMD machines have a front-end processor for handling the code and the global operations. Behind the front-end processor is the processor array

consisting of sometimes thousands of processors. In most cases, the interconnection network of the processor array is a mesh (or grid). In a SIMD environment, processors work synchronously, that is, in lock step. Data are partitioned across the processors, and each processor performs the same instruction on its data. Process coordination is handled in the hardware, and process communication is usually handled with built-in primitives. In a SIMD program, multiple processors execute the same process at the same time. Since each processor has different data, it is the data transformations that are being performed in parallel. Thus, this form of parallelism is usually called data (or fine-grained) parallelism.

The SIMD environment is perfect for the diffusion model. However, if the only available environment is a MIMD environment, it is still possible to implement data parallelism, and, thus, the diffusion model. In the diffusion model, it is not necessary for the evolution of the individual to take place in lock step. Each individual can evolve independently of the others much as in nature. Therefore, it is possible to use the SPMD programming paradigm. In a SPMD environment, each processor runs the same program on different data. Because the underlying hardware is MIMD, the processes will run asynchronously. The programmer is responsible for handling process coordination and communication using locks, barriers, shared variables, or message passing. In this situation, the easiest solution would be to allow the individuals to evolve asynchronously. However, using barriers, it is possible to force the processes to wait on all other processes before proceeding with the next generation.

With a basic understanding of data parallelism, it is now possible to continue with the discussion of the implementation issues in the diffusion model. For additional information on functional parallelism, data parallelism, MIMD machines or environments, or SIMD machines see the books by Almasi and Gottlieb (1994) or Morse (1994).

C6.4.2.2 Techniques for selecting parents

No matter what technique is used for selecting the parents, it will be necessary for each process to communicate with some (maybe all) of its neighbors. If possible, communication in a parallel environment should be kept to a minimum in order to improve the performance of the algorithm. Therefore, the neighborhood sizes are usually kept small in order to alleviate the communication problem.

Quite frequently the technique for selecting the parents for recombination is a local version of a standard global selection technique. When converting a global technique to a local technique it is simply a matter of implementing the global technique as if it were in a much smaller population. This will mean collecting *performance measures* from all individuals in the neighborhood, or as in the case of *tournament selection* collecting the performance measures from a random set of individuals in the neighborhood. B2.4, C2.3

There are many examples of local implementations of global techniques. For example, Gorges-Schleuter (1989), Manderick and Spiessens (1989), and Davidor (1991) all use *proportional selection* for at least one parent. The difference between the three techniques is in the selection of the second parent. Gorges-Schleuter chooses the process' individual as the second parent. Manderick and Spiessens choose the second parent randomly. Davidor chooses both parents based on the probability distribution of the neighborhood fitnesses. C2.2

Other examples of selection techniques in diffusion model implementations may involve as much communication as the previously mentioned techniques. For example, Farrell *et al* (1994) have devised one such technique. The first parent chosen is the individual. The second parent is the most successful individual in the neighborhood.

One technique which may or may not involve as much communication is the technique devised by Collins and Jefferson (1991). In their diffusion model, each parent is selected by performing a random walk. The fittest individual found in the random walk becomes the parent. Of course the length of the random walk determines the amount of communication.

Probably the most unique technique is to choose all of the neighbors as parents. Mühlenbein (1989) chose the four neighbors, the individual, and the global best individual was chosen twice (i.e. there were seven parents!). Of course this type of selection affects the recombination. Recombination techniques are discussed in the following section.

C6.4.2.3 Recombination techniques

In most cases the recombination technique is a 'typical' technique. In other words, the technique is the same as a sequential EA. One interesting deviation from the typical techniques is the p-sexual voting

(Mühlenbein 1989). As was mentioned previously, in this diffusion model, each child has more than two parents. All parents vote on which allele should be chosen for a particular gene. If one allele receives more than some threshold number of votes, then that allele wins. Otherwise an allele is chosen randomly for the gene.

Regardless of the recombination technique, the number of children created by recombination can be one or two as in sequential EAs. Also as in sequential EAs, the question arises as to what should be done with the child(ren). If only one child is created, it usually replaces the individual, although Gorges-Schleuter (1989) only allows a replacement to occur if the fitness of the child is not the worst fitness in the neighborhood. If two children are created, usually one is chosen (see e.g. Manderick and Spiessens 1989) and it replaces the individual. However, Davidor (1991) creates two children and places both in the neighborhood. If the child is different from the individual which it is supposed to replace, then one of the two is selected based on the probability distribution created by their two fitnesses.

Often the selection technique and the replacement of children is influenced by the size and shape of the neighborhood. Some typical deme attributes are presented in the following section.

C6.4.2.4 Deme attributes: size and shape

In most diffusion model implementations, the size and shape of the deme is influenced by the underlying architecture. For instance, the most common SIMD interconnection topology is a mesh or grid. Given this underlying architecture, the typical selection for a neighborhood is the neighboring processors on the grid. In many of these machines it is also possible to quickly communicate with the NE, NW, SE, and SW neighbors as well as the neighbors immediately above, below, to the left, and to the right of the processor. This creates a neighborhood of nine individuals. For example, in figure C6.4.1 the circles represent processors, and the lines represent connections between processors. The neighborhood of the individual residing on the processor represented by the open circle would be all nine circles in the figure (Manderick and Spiessens 1989, Davidor 1991).

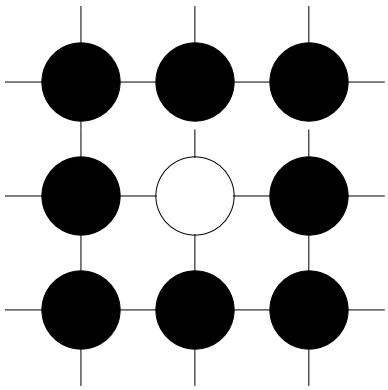


Figure C6.4.1. A mesh (or grid) neighborhood.

The underlying architecture used by Mühlenbein (1989) and Gorges-Schleuter (1989) was a double ring. Each processor in each ring was connected to exactly one processor in the other ring. This produced a ladder-like topology. In this situation, an individual's neighborhood can be determined by how close a processor is to other processors on the ladder. Closeness is usually defined in terms of how many communication links away a processor is from a given processor. For instance, if the neighborhood is defined as being all processors no more than one link away, then the neighborhood is T shaped with the individual, the individual to the left on the ring, the individual to the right on the ring, and the neighboring individual on the other ring (see figure C6.4.2). Figure C6.4.3 shows the neighborhood if the distance between neighbors is two.

Occasionally the selection technique and not the architecture affects the size and shape of the neighborhood. Collins and Jefferson (1991) implemented their diffusion model on a grid, but because the selection technique was a random walk, then the selection technique affected the deme shape and size. The size of the deme was the length of the random walk. The shape of the deme was random.

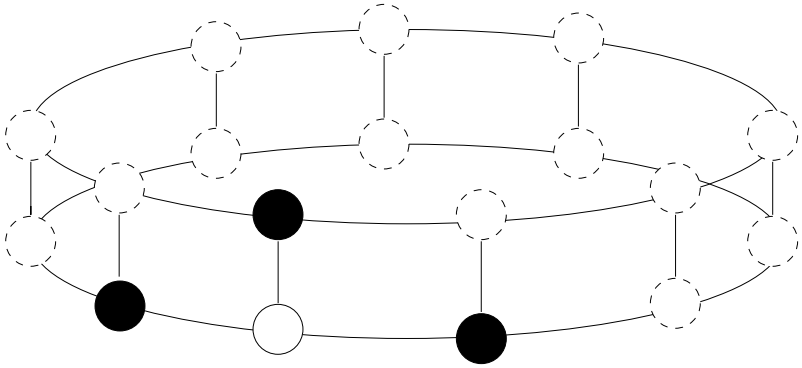


Figure C6.4.2. A ladder neighborhood with a distance of one.

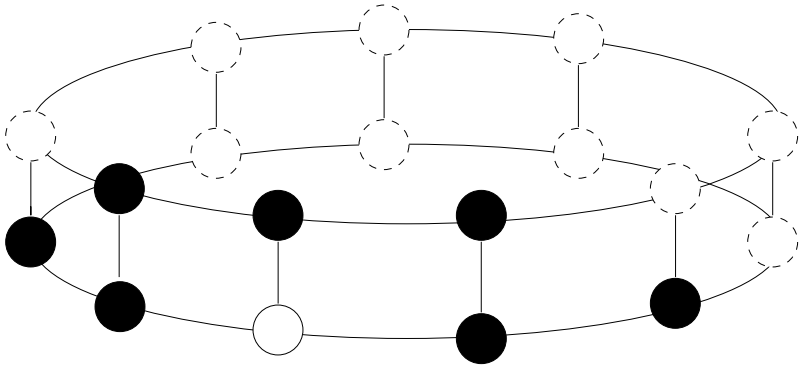


Figure C6.4.3. A ladder neighborhood with a distance of two.

In this section several implementation techniques have been presented. It should be noted that most techniques cause a theoretical deviation from the underlying sequential EA theory. In the next section a few remarks are made about theoretical research in the area of diffusion models.

C6.4.3 Theoretical research in diffusion models

Very little has been done theoretically in the area of diffusion models. This is probably due in part to the difficulty of deriving generic proofs in an area where there are so many different implementation possibilities. Another possibility for the lack of theory may be the belief that the diffusion model more correctly models natural populations than the island model or sequential EAs. Regardless of the reason for the lack of theory, more work needs to be done in this area. Below are mentioned three of the theoretical results that have been published.

Davidor (1991) derived a schema theorem for eight neighbors on a grid. The theorem was based on using proportional selection for both parents, creating two children, and proportionally placing both children in the neighborhood. Using this theorem, he found that the diffusion model had a rapid but local convergence creating 'islands of near optimal strings'. This rapid, local convergence is no surprise considering that the selection is effectively in very small populations.

Spiessens and Manderick (1991) performed a comparison of the time complexity of their diffusion model and a sequential GA. They ignored the evaluation step since this is problem dependent. They were able to show that the complexity of the diffusion model increases linearly with respect to the length of the genotype. The complexity of a sequential GA increases polynomially with respect to the size of the population multiplied by the length of the genotype. Since, theoretically, an increase in the length of an individual should be accompanied by an increase in the population size, an increase in the length of an individual will affect the run time of a sequential GA, but it will not affect the run time of the diffusion model. Also in this article, they derive the expected number of individuals due to proportional,

scaling, local ranking, and local tournament selection. The growth rates are then compared showing that proportional selection has the lowest growth rate.

The final result presented here was actually a result of experiments done by De Jong and Sarma (1995). In their work they discovered a result that needs to be kept in mind by all who would implement the diffusion model. Their experiments compared proportional, ranking, and binary tournament selection. While performing their experiments they found that binary tournament selection appeared to perform worse than linear ranking. This was surprising given that the two techniques have equivalent selection pressures. 'These results emphasize the importance of an analysis of the variance of selection schemes. Without it one can fall into the trap of assuming that selection algorithms that have equivalent expected selection pressure produce similar search behavior' (De Jong and Sarma 1995).

C6.4.4 Conclusion

The diffusion model is perhaps the most 'natural' EA in that it seems to simulate the evolution of natural populations from the point of view of the individual. While there are a few implementation issues that are unique to the diffusion model, it is, none the less, a fairly simple algorithm to implement. This, coupled with the increasing availability of SIMD and SPMD environments, makes the diffusion model an excellent choice for improving the search of an EA.

C6.4.5 Additional sources of information

It was impossible to list all the authors or all the implementations of the diffusion model. Therefore, to avoid accidentally leaving someone out, only a few examples were chosen for the preceding sections and the bibliography. There are also many good Internet resources, but most of them can be reached from ENCORE (the evolutionary computation repository) on the World Wide Web.

References

Almasi G S and Gottlieb A 1994 *Highly Parallel Computing* (Redwood City, CA: Benjamin–Cummings)

Collins R J and Jefferson D R 1991 Selection in massively parallel genetic algorithms *Proc. 4th Int. Conf. on Genetic Algorithms (University of California, San Diego, CA, July 1991)* ed R K Belew and L B Booker (San Mateo, CA: Morgan Kaufmann) pp 249–56

Davidor Y 1991 A naturally occurring niche and species phenomenon: the model and first results *Proc. 4th Int. Conf. on Genetic Algorithms (University of California, San Diego, CA, July 1991)* ed R K Belew and L B Booker (San Mateo, CA: Morgan Kaufmann) pp 257–63

De Jong K and Sarma J 1995 On decentralizing selection algorithms *Proc. 6th Int. Conf. on Genetic Algorithms (Pittsburgh, PA, July 1995)* ed L J Eshelman (San Mateo, CA: Morgan Kaufmann) pp 17–23

Farrell C A, Kieronska D H and Schulze M 1994 Genetic algorithms for network division problem *Proc. 1st IEEE Conf. on Evolutionary Computation (Orlando, FL, June 1994)* (Piscataway, NJ: IEEE) pp 422–7

Flynn M J 1966 Very high speed computing systems *Proc. IEEE* **54** 1901–9

Gorges-Schleuter M 1989 ASPARAGOS: an asynchronous parallel genetic optimization strategy *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, June 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 422–7

Hartl D L 1980 *Principles of Population Genetics* (Sunderland, MA: Sinauer)

Manderick B and Spiessens P 1989 Fine-grained parallel genetic algorithms *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, June 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 428–33

Morse H S 1994 *Practical Parallel Computing* (Cambridge, MA: AP Professional)

Mühlenbein H 1989 Parallel genetic algorithms, population genetics and combinatorial optimization *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, June 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 416–21

Spiessens and Manderick 1991 A massively parallel genetic algorithm: implementation and first analysis *Proc. 4th Int. Conf. on Genetic Algorithms (University of California, San Diego, CA, July 1991)* ed R K Belew and L B Booker (San Mateo, CA: Morgan Kaufmann) pp 279–86

Whitley D 1993 Cellular genetic algorithms *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann)

C7.1 Self-adaptation

Thomas Bäck

Abstract

The principle of self-adaptation, as mainly utilized in evolution strategies and evolutionary programming, facilitates the implicit control of strategy parameters by incorporating them into the representation of individuals and by evolving the strategy parameters themselves in analogy with the usual evolution of object variables. This section provides an overview of a number of existing techniques for the self-adaptation of control parameters related to the mutation and recombination operators in evolutionary algorithms and illustrates the strengths of this approach by several simple experiments. It is demonstrated that self-adaptation works under a variety of conditions regarding the search space of the underlying optimization problem (i.e. continuous, binary, integer, and finite-state machine spaces) and choices of the probability density function used for the probabilistic variation of strategy parameters. Though a number of open questions remain, the principle of self-adaptation (which has a natural model in repair enzymes and mutator genes that in part control the DNA mutation rate of mammals) is identified as a general, robust, and efficient mechanism for parameter control in evolutionary algorithms.

C7.1.1 Introduction

The *self-adaptation* of strategy parameters provides one of the key features of the success of *evolution strategies* and *evolutionary programming*, because both evolutionary algorithms use evolutionary principles to search in the space of object variables and strategy parameters simultaneously. B1.3, B1.4

The term *strategy parameters* refers to parameters that control the evolutionary search process, such as mutation rates, mutation variances, and recombination probabilities, and the idea of self-adaptation consists in evolving these parameters in analogy to the object variables themselves. Typically, strategy parameters are self-adapted on the level of individuals, by incorporating them into the representation of individuals in addition to the set of object variables; that is, the individual space I is given by

$$I = A_x \times A_s \tag{C7.1.1}$$

where A_x denotes the set of object variables (i.e. of representations of solutions) and A_s denotes the set of strategy parameters.

For an individual $\boldsymbol{a} = (\boldsymbol{x}, \boldsymbol{s})$ consisting of an object variable vector $\boldsymbol{x}$ and a strategy parameter set $\boldsymbol{s}$, the self-adaptation mechanism is typically implemented by first (recombining and) mutating (according to some probability density function) the strategy parameter vector $\boldsymbol{s}$, yielding $\boldsymbol{s}'$, and then using the updated strategy parameters $\boldsymbol{s}'$ to (recombine and) mutate the object variable vector $\boldsymbol{x}$, yielding $\boldsymbol{x}'$.

Consequently, rather than using some deterministic control rule for the modification of strategy parameters, they are themselves subject to evolutionary operators and probabilistic changes. Selection is still performed on the basis of the objective function value $f(\boldsymbol{x})$ only; that is, strategy parameters are selected for survival by means of the *indirect link* between strategy parameters and the objective function value. Since the mechanism works on the basis of rewarding improvements in objective function value, strategy parameters are continuously adapted such that convergence velocity is emphasized by the

evolutionary algorithm, but the speed of the adaptation on the level of strategy parameters is under the control of the user by means of so-called *learning rates*.

It should be noted that the self-adaptation principle is fundamentally different from other parameter control mechanisms for evolutionary algorithms such as *dynamic parameter control* or *adaptive parameter control*—a classification that was recently proposed by Eiben and Michalewicz (1996). Under dynamic parameter control, the parameter settings obtain different values according to a deterministic schedule prescribed by the user. An overview of dynamic schedules can be found in Section E1.2. Adaptive E1.2 parameter control mechanisms obtain new values by a feedback mechanism that monitors evolution and explicitly rewards or punishes operators according to their impact on the objective function value. Examples of this mechanism are the method of Davis (1989) to adapt operator probabilities in genetic algorithms based on their observed success or failure to yield a fitness improvement and the approaches of Arabas *et al* (1994) and Schlierkamp-Voosen and Mühlenbein (1996) to adapt population sizes either by assigning lifetimes to individuals based on their fitness or by having a competition between subpopulations based on the fitness of the best population members. In contrast to these approaches, *self-adaptive parameter control* works by encoding parameters in the individuals and evolving the parameters themselves.

The following sections give an overview of some of the approaches for self-adaptation of strategy parameters described in the literature.

C7.1.2 Mutation operators

Most of the research and successful applications of self-adaptation principles in evolutionary algorithms deal with parameters related to the mutation operator. The technique of self-adaptation is most widely utilized for the variances and covariances of a generalized n-dimensional normal distribution, as introduced by Schwefel (1977) in the context of evolution strategies and Fogel (1992) for the parameter optimization variants of evolutionary programming. The case of continuous object variables $x_i \in \mathbb{R}$ motivated a number of successful recent attempts to transfer the method to other search spaces such as binary vectors, discrete spaces in general, and even finite-state machines. In the following subsections, the corresponding self-adaptation principles are described in some detail.

C7.1.2.1 Continuous search spaces

In the most general case, an individual $\boldsymbol{a} = (\boldsymbol{x}, \boldsymbol{\sigma}, \boldsymbol{\alpha})$ of a (μ, λ) evolution strategy consists of up to three components $\boldsymbol{x} \in \mathbb{R}^n$, $\boldsymbol{\sigma} \in \mathbb{R}^{n_\sigma}$, and $\boldsymbol{\alpha} \in [-\pi, \pi]^{n_\alpha}$, where $n_\sigma \in \{1, \ldots, n\}$ and $n_\alpha \in \{0, (2n - n_\sigma)(n_\sigma - 1)/2\}$. The mutation operator works by adding a realization of a normally distributed n-dimensional random variable $\boldsymbol{X} \sim \mathbf{N}(\mathbf{0}, \mathbf{C})$ with expectation vector $\mathbf{0}$, covariance matrix

$$\mathbf{C} = (c_{ij}) = \begin{cases} \operatorname{cov}(X_i, X_j) & i \neq j \\ \operatorname{var}(X_i) & i = j \end{cases} \tag{C7.1.2}$$

and probability density function

$$f_{\boldsymbol{X}}(x_1, \ldots, x_n) = \frac{\exp\left(-\frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\mathbf{C}^{-1}\boldsymbol{x}\right)}{((2\pi)^n \det(\mathbf{C}))^{1/2}} \tag{C7.1.3}$$

where the covariance matrix is described by the mutated strategy parameters $\boldsymbol{\sigma}'$ and $\boldsymbol{\alpha}'$ of the individual. Depending on the number of strategy parameters incorporated into the representation of an individual, the following main variants of self-adaptation can be distinguished.

(i) $n_\sigma = 1$, $n_\alpha = 0$, $\boldsymbol{X} \sim \sigma'\mathbf{N}(\mathbf{0}, \mathbf{I})$. The standard deviation for all object variables is identical (σ'), and all object variables are mutated by adding normally distributed random numbers with

$$\sigma' = \sigma \exp(\tau_0 N(0, 1)) \tag{C7.1.4}$$

$$x_i' = x_i + \sigma' N_i(0, 1) \tag{C7.1.5}$$

where $\tau_0 \propto n^{-1/2}$ and $N_i(0, 1)$ denotes a realization of a one-dimensional normally distributed random variable with expectation zero and standard deviation one that is sampled anew for each index i. The lines of equal probability density of the normal distribution are hyperspheres in this case, as shown graphically for $n = 2$ in the left-hand part of figure C7.1.1.

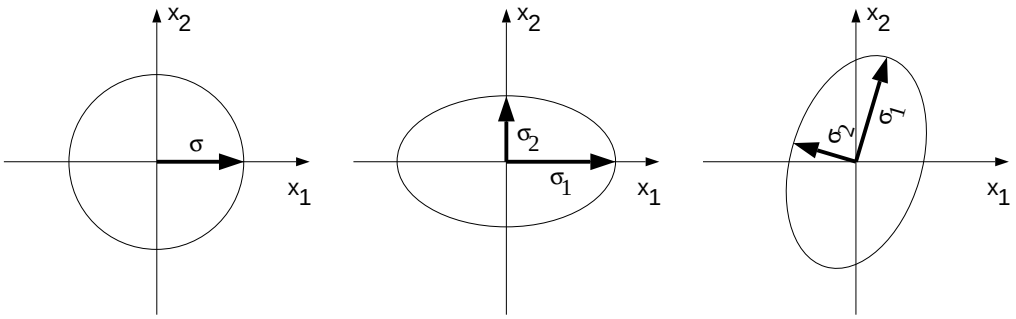


Figure C7.1.1. A sketch of the lines of equal probability density of the $n = 2$-dimensional normal distribution in the case of simple mutations with $n_\sigma = 1$ (left), $n_\sigma = 2$ (middle), and correlated mutations with $n_\sigma = 2$, $n_\alpha = 1$ (right).

(ii) $n_\sigma = n$, $n_\alpha = 0$, $\boldsymbol{X} \sim \mathbf{N}(\mathbf{0}, \boldsymbol{\sigma}'\mathbf{I})$. All object variables have their own, individual standard deviations σ_i, which determine the corresponding modifications according to

$$\sigma_i' = \sigma_i \exp(\tau' N(0,1) + \tau N_i(0,1)) \tag{C7.1.6}$$

$$x_i' = x_i + \sigma_i' N_i(0,1) \tag{C7.1.7}$$

where $\tau' \propto (2n)^{-1/2}$ and $\tau \propto (2n^{1/2})^{-1/2}$. The lines of equal probability density of the normal distribution are hyperellipsoids, as shown in the middle part of figure C7.1.1 for $n = 2$.

(iii) $n_\sigma = n$, $n_\alpha = n(n-1)/2$, $\boldsymbol{X} \sim \mathbf{N}(\mathbf{0}, \mathbf{C})$. The vectors $\boldsymbol{\sigma}$ and $\boldsymbol{\alpha}$ represent the complete covariance matrix of the n-dimensional normal distribution, where the covariances c_{ij} ($i \in \{1, \ldots, n-1\}$, $j \in \{i+1, \ldots, n\}$) are represented by a vector of rotation angles α_k ($k = \frac{1}{2}(2n-i)(i+1) - 2n + j$) describing the coordinate rotations necessary to transform an uncorrelated mutation vector into a correlated one. Rotation angles and covariances are related to each other according to

$$\tan(2\alpha_k) = \frac{2c_{ij}}{\sigma_i^2 - \sigma_j^2}. \tag{C7.1.8}$$

By using the rotation angles to represent the covariances, the mutation operator is guaranteed to generate exactly the feasible (positive definite) covariance matrices and to allow for the creation of any possible covariance matrix (for details see the article by Rudolph (1992)). The mutation is performed according to

$$\sigma_i' = \sigma_i \exp(\tau' N(0,1) + \tau N_i(0,1)) \tag{C7.1.9}$$

$$\alpha_j' = \alpha_j + \beta N_j(0,1) \tag{C7.1.10}$$

$$\boldsymbol{x}' = \boldsymbol{x} + \mathbf{N}(\mathbf{0}, \mathbf{C}(\boldsymbol{\sigma}', \boldsymbol{\alpha}')) \tag{C7.1.11}$$

where $\mathbf{N}(\mathbf{0}, \mathbf{C}(\boldsymbol{\sigma}', \boldsymbol{\alpha}'))$ denotes the correlated mutation vector and $\beta \approx 0.0873$ (5°). As shown in the right-hand part of figure C7.1.1 for $n = 2$, the mutation hyperellipsoids are now arbitrarily rotatable, and α_k (with $k = \frac{1}{2}(2n-i)(i+1) - 2n + j$) characterizes the rotation angle with respect to the coordinate axes i and j.

(iv) $1 < n_\sigma < n$. The general case of having neither just one nor the full number of different degrees of freedom available is also permitted, and implemented by the agreement to use σ_{n_σ} for mutating all x_i where $n_\sigma \leq i \leq n$.

The settings for the *learning rates* τ, τ', and τ_0 are recommended by Schwefel as reasonable heuristic settings (see Schwefel 1977, pp 167–8), but one should have in mind that, depending on the particular topological characteristics of the objective function, the optimal setting of these parameters might differ from the values proposed. For $n_\sigma = 1$, however, Beyer (1995b) has recently theoretically shown that, for the sphere model

$$f(\boldsymbol{x}) = \sum_{i=1}^{n} (x_i - x_i^*)^2 \tag{C7.1.12}$$

the setting $\tau_0 \propto n^{-1/2}$ is the optimal choice, maximizing the convergence velocity of the evolution strategy. Moreover, for a $(1, \lambda)$ evolution strategy Beyer derived the result that $\tau_0 \approx c_{1,\lambda}/n^{1/2}$ (for $\lambda \geq 10$), where $c_{1,\lambda}$ denotes the *progress coefficient* of the $(1, \lambda)$ strategy. B2.4

For an empirical investigation of the self-adaptation mechanism defined by the mutation operator variants (i)–(iii), Schwefel (1987, 1989, 1992) used the following three objective functions which are specifically tailored to the number of learnable strategy parameters in these cases.

(i) Function

$$f_1(\boldsymbol{x}) = \sum_{i=1}^{n} x_i^2 \tag{C7.1.13}$$

requires learning of one common standard deviation σ, i.e. $n_\sigma = 1$.

(ii) Function

$$f_2(\boldsymbol{x}) = \sum_{i=1}^{n} i x_i^2 \tag{C7.1.14}$$

requires learning of a suitable *scaling* of the variables, i.e. $n_\sigma = n$.

(iii) Function

$$f_3(\boldsymbol{x}) = \sum_{i=1}^{n} \left(\sum_{j=1}^{i} x_j \right)^2 \tag{C7.1.15}$$

requires learning of a positive definite *metrics*, i.e. individual σ_i and $n_\alpha = n(n-1)/2$ different covariances.

As a first experiment, Schwefel compared the convergence velocity of a $(1, 10)$ and a $(1 + 10)$ evolution strategy with $n_\sigma = 1$ on the sphere model f_1 with $n = 30$. The results of a comparable experiment performed by the present author (averaged over ten independent runs, with the standard deviations initialized with a value of 0.3) are shown in figure C7.1.2 (left), where the convergence velocity or progress is measured by $\log((f_{\min}(0)/f_{\min}(g))^{1/2})$ with $f_{\min}(g)$ denoting the objective function value in generation g. It is somewhat counterintuitive to observe that the nonelitist $(1, 10)$ strategy, where all offspring individuals might be worse than the single parent, performs *better* than the elitist $(1 + 10)$ strategy. This can be explained, however, by taking into account that the self-adaptation of standard deviations might generate an individual with a good objective function value but an inappropriate value of σ for the next generation. In the case of a plus strategy, this inappropriate standard deviation might survive for a number of generations, thus hindering the combined process of search and adaptation. The resulting periods of stagnation can be prevented by allowing the good search point to be forgotten, together with its inappropriate step size. From this experiment, Schwefel concluded that the nonelitist (μ, λ) selection mechanism is an important condition for a successful self-adaptation of strategy parameters. Recent experimental findings by Gehlhaar and Fogel (1996) on objective functions more complicated than the sphere model give some evidence, however, that the elitist strategy performs as well as or even better than the (μ, λ) strategy in many practical cases.

For a further illustration of the self-adaptation principle in case of the sphere model f_1, we use a time-varying version where the optimum location $\boldsymbol{x}^* = (x_1^*, \ldots, x_n^*)$ is changed every 150 generations. Ten independent experiments for $n = 30$ and 1000 generations per experiment are performed with a $(15, 100)$ evolution strategy (without recombination). The average best objective function value (solid curve) and the minimum, average, and maximum standard deviations $\sigma_{\min}$, σ_{avg}, and $\sigma_{\max}$ are shown in the right-hand part of figure C7.1.2. The curve of the objective function value clearly illustrates the linear convergence of the algorithm during the first search interval of 150 generations. After shifting the optimum location at generation 150, the search stagnates for a while at the bad new position before the linear convergence is observed again.

The behavior of the standard deviations, which are also plotted in figure C7.1.2 (right), clarifies the reason for the periods of stagnation of the objective function values: self-adaptation of standard deviations works both by decreasing them during the periods of linear convergence and by increasing them during the periods of stagnation, back to a magnitude such that they have an impact on the objective function value. This process of standard deviation increase, which occurs at the beginning of each interval, needs

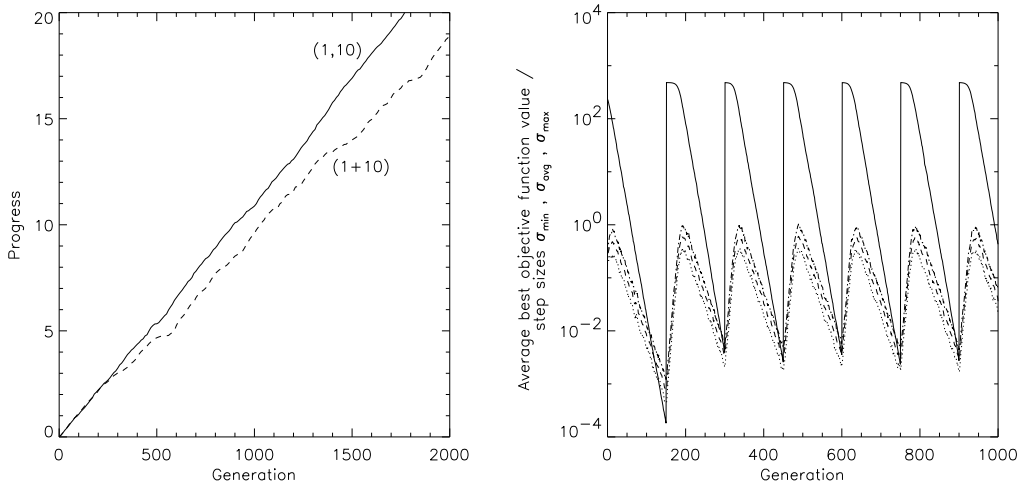


Figure C7.1.2. Left: a comparison of the convergence velocity of a $(1, 10)$ strategy and a $(1+10)$ strategy in the case of the sphere model f_1 with $n = 30$ and $n_\sigma = 1$. Right: the best objective function value and minimum, average, and maximum standard deviation in the population plotted over the generation number for the time-varying sphere model. The results were obtained by using a $(15, 100)$ evolution strategy with $n_\sigma = 1$, $n = 30$, without recombination.

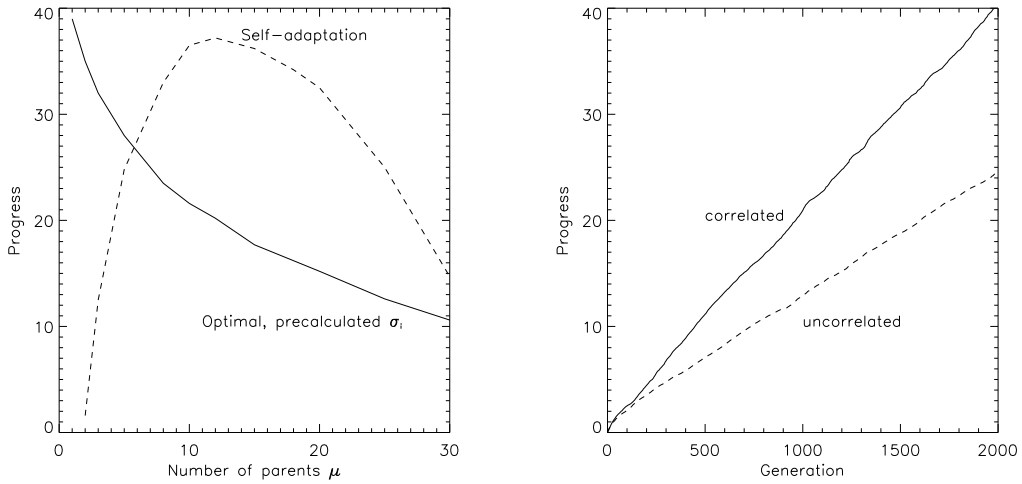


Figure C7.1.3. Left: the convergence velocity on f_2 for a $(\mu, 100)$ strategy with $\mu \in \{1, \ldots, 30\}$ for the self-adaptive evolution strategy and the strategy using optimum prefixed values of the standard deviations σ_i. Right: a comparison of the convergence velocity of a $(15, 100)$ strategy with correlated mutations in the case of the function f_3 with $n = n_\sigma = 10$, $n_\alpha = 45$ and with self-adaptation of standard deviations only (uncorrelated) for $n = n_\sigma = 10, n_\alpha = 0$.

some time which does not yield any progress with respect to the objective function value. According to Beyer (1995b), the number of generations needed for this adaptation is inversely proportional to τ_0^2 (that is, proportional to n) in the case of a $(1, \lambda)$ evolution strategy.

In the case of the objective function f_2, each variable x_i is differently scaled by a factor $i^{1/2}$, such that self-adaptation requires the scaling of n different σ_i to be learned. The optimal settings of standard deviations $\sigma_i^* \propto i^{-1/2}$ are also known in advance for this function, such that self-adaptation can be compared to an evolution strategy using optimally adjusted σ_i for mutation. The result of this comparison is shown in figure C7.1.3 (left), where the convergence velocity is plotted for $(\mu, 100)$ evolution strategies as a function of μ, the number of parents, for both the self-adaptive strategy and the strategy using the optimal setting of σ_i.

It is not surprising to see that, for the strategy using optimal standard deviations σ_i, the convergence rate is maximized for $\mu = 1$, because this setting exploits the perfect knowledge in an optimal sense. In the case of the self-adaptive strategy, however, a clear maximum of the progress rate is reached for a value of $\mu = 12$, and both larger and smaller values of μ cause a strong loss of convergence speed. The collective performance of about 12 imperfect parents, achieved by means of self-adaptation, is almost equal to the performance of the perfect $(1, 100)$ strategy and outperforms the collection of 12 perfect individuals by far. This experiment indicates that self-adaptation is a mechanism that requires the existence of a knowledge diversity (or diversity of internal models), i.e. a number of parents larger than one, and benefits from the phenomenon of collective (rather than individual) intelligence.

Concerning the objective function f_3, figure C7.1.3 (right) shows a comparison of the progress for a $(15, 100)$ evolution strategy with $n_\sigma = n = 10$, $n_\alpha = 0$ (that is, no correlated mutations) and $n_\alpha = n(n-1)/2 = 45$ (that is, full correlations). In both cases, *intermediary recombination* of object variables, global intermediary recombination of standard deviations, and no recombination of the rotation angles is chosen. The results demonstrate that, by introducing the covariances, it is possible to increase the effectiveness of the collective learning process in case of arbitrarily rotated coordinate systems. Rudolph (1992) has shown that an approximation of the Hessian matrix could be computed by correlated mutations with an upper bound of $\mu + \lambda = (n^2 + 3n + 4)/2$ on the population size, but the typical settings ($\mu = 15$, $\lambda = 100$) are often not sufficient to achieve this (an experimental investigation of the scaling behavior of correlated mutations with increasing population sizes and problem dimension has not yet been performed). C3.3.2

The choice of a logarithmic normal distribution for the modification of the standard deviations σ_i in connection with a multiplicative scheme in equations (C7.1.4), (C7.1.6), and (C7.1.9) is motivated by the following heuristic arguments (see Schwefel 1977, p 168):

(i) A multiplicative process preserves positive values.
(ii) The median should be equal to one to guarantee that, on average, a multiplication by a certain value occurs with the same probability as a multiplication by the reciprocal value (i.e. the process would be neutral under the absence of selection).
(iii) Small modifications should occur more often than large ones.

The effectiveness of this multiplicative logarithmic normal modification is presently also acknowledged in evolutionary programming, since extensive empirical investigations indicate some advantage of this scheme over the original additive self-adaptation mechanism used in evolutionary programming (Saravanan 1994, Saravanan and Fogel 1994, Saravanan *et al* 1995), where

$$\sigma_i' = \sigma_i(1 + \alpha N(0, 1)) \tag{C7.1.16}$$

(with a setting of $\alpha \approx 0.2$ (Saravanan *et al* 1995)). Recent investigations indicate, however, that this becomes reversed when noisy objective functions are considered, where the additive mechanism seems to outperform multiplicative modifications (Angeline 1996).

The study by Gehlhaar and Fogel (1996) also indicates that the order of the modifications of x_i and σ_i has a strong impact on the effectiveness of self-adaptation: it is important to mutate the standard deviations first and to use the mutated standard deviations for the modification of object variables. As the authors point out in that study, the reversed mechanism might suffer from generating offspring that have useful object variable vectors but bad strategy parameter vectors, because these have not been used to determine the position of the offspring itself.

Concerning the sphere model f_1 and a $(1, \lambda)$ strategy, Beyer (1995b) has recently indicated that equation (C7.1.16) is obtained from equation (C7.1.4) by Taylor expansion breaking off after the linear term, such that both mutation mechanisms should behave identically for small settings of the learning rates τ_0 and α, when $\tau_0 = \alpha$. This was recently confirmed by Bäck and Schwefel (1996) with some experiments for the time-varying sphere model. Moreover, Beyer (1995b) also shows that the self-adaptation principle works for a variety of different probability density functions for the modification of step sizes; that is, it is a very robust technique. For $n_\sigma = 1$, even the simple mutational step size control

$$\sigma' = \begin{cases} \sigma\alpha & \text{if } u \sim U(0,1) \leq \frac{1}{2} \\ \sigma/\alpha & \text{if } u \sim U(0,1) > \frac{1}{2} \end{cases} \tag{C7.1.17}$$

of Rechenberg (1994, p 47) provides a reasonable choice. A value of $\alpha = 1.3$ of the learning rate is proposed by Rechenberg.

In concluding this subsection, recent approaches to substituting the normal distribution used for the modification of object variables x_i by other probability densities are worth mentioning. As outlined by Yao and Liu (1996), the one-dimensional Cauchy density function

$$f(x) = \frac{1}{\pi} \frac{t}{t^2 + (x-u)^2} \tag{C7.1.18}$$

is a good candidate, because its shape resembles that of the Gaussian density function, but approaches the axis so slowly that expectation (and higher moments) do not exist. Consequently, it is natural to hope that the Cauchy density increases the probability of leaving local optima. Because the moments of a one-dimensional Cauchy distribution do not exist, Yao and Liu (1996) use the same self-adaptation mechanism as described by equations (C7.1.6) and (C7.1.7) with the only modification to substitute the standardized normally distributed $N(0,1)$ in equation (C7.1.7) by a random variable with one-dimensional Cauchy distribution with parameters $u = 0$, $t = 1$. A realization of such a random variable can be generated by dividing the realizations of two independent standard normally distributed random variables.

Using a large set of 23 test functions, the authors of this study conclude that their new algorithm (called *fast evolutionary programming*) performs better than the implementation using the normal rather than Cauchy distribution especially for multimodal functions with many local optima while being comparable to the normal distribution for unimodal and multimodal functions with only a few local optima.

Finally, the most general variant of self-adaptation seems to consist in the self-adaptation of the whole probability density function itself rather than having a fixed density and adapting one or more control parameters of that density. Davis (1994) implements this idea by representing a one-dimensional continuous probability density function by a discrete mutation histogram with 101 bars in width over a region of interest $[a, b]$. The heights of the histogram bars are integer values $h_0 h_1 \ldots h_{101}$, the histogram specifies a region that ranges from $x-(b-a)/2$ to $x+(b-a)/2$ around the current value x of a solution, and h_{50} is centered at the current value of the solution.

In Davis' implementation, an object variable is mutated by choosing a new solution value from the probability density function over the histogram's range. Afterwards, the mutation density is modified by choosing a histogram bar using the same probability density function, and incrementing the bar value with a probability proportional to the bar height. Using this mechanism for self-adaptation of the probability density function itself, Davis found that for landscapes with local optima the shape of the probability density function is adapted to reflect the landscape structure with respect to the location of the optima. Moreover, the formation of a peaked center in each of the mutation histograms is interpreted by Davis as a hint that the normal distribution naturally emerges as a good choice for a wide range of fitness landscapes.

While it is not surprising to see from these experiments that the structure of a one-dimensional objective function can be learned by self-adaptation, it is necessary here to emphasize that the extension of this approach to $n \gg 1$ dimensions fails because the discretized representation of an arbitrary multivariate distribution would require c^n histogram bars (c being the number of bars in one dimension). The condensed representation of a multivariate distribution by a few control parameters which are self-adapted (as in equations (C7.1.9)–(C7.1.11)) is a more appropriate method to handle the higher-dimensional case than the representation suggested by Davis.

C7.1.2.2 Binary search spaces

A transfer of the self-adaptation principle from evolution strategies to the mutation probability $p_m \in [0,1]$ of canonical (i.e. with a binary representation of individuals) *genetic algorithms* was first proposed by Bäck (1992b). Based on the binary representation $\boldsymbol{b} = (b_1 \ldots b_\ell) \in \{0,1\}^\ell$ of object variables (often, continuous object variables $x_i \in [u_i, v_i] \subset \mathbb{R}$, $i = 1, \ldots, n$, are represented in canonical genetic algorithms by binary strings and a Gray code; see C4.2 for details), Bäck extended the representation by additional ℓ' (or $n\ell'$) bits to encode either one or n mutation probabilities (the latter can only be applied if the genotype $\boldsymbol{b}$ explicitly splits into n logical subparts encoding different object variables) as part of each individual. Because of the restricted applicability of the general case of n mutation probabilities, we discuss only the case of one mutation probability here.

B1.2
C4.2

An individual $\boldsymbol{a} = (\boldsymbol{b}, \boldsymbol{p})$ consists of the binary vector $\boldsymbol{b} = (b_1 \ldots b_\ell) \in \{0,1\}^\ell$ representing the object variables and the binary vector $\boldsymbol{p} = (p_1 \ldots p_{\ell'}) \in \{0,1\}^{\ell'}$ representing the individual's mutation rate p_m

according to the decoding function $\Gamma_{0,1,\ell'}$, which is defined as follows:

$$\Gamma_{u,v,\ell}(b_1 \ldots b_\ell) = u + (v-u)\frac{\sum_{i=0}^{\ell-1}\left(\bigoplus_{j=1}^{i+1} b_j\right)2^i}{2^\ell - 1}. \tag{C7.1.19}$$

Here, $\oplus$ denotes summation modulo 2, such that a Gray code and a linear mapping of the decoded integer to the range $[u, v]$ are used. With the definition given in equation (C7.1.19), p_m and $\boldsymbol{p}$ are related by $p_\mathrm{m} = \Gamma_{0,1,\ell'}(\boldsymbol{p})$, and the mutation operator for self-adapting p_m proceeds by mutating $\boldsymbol{p}$ with mutation rate p_m, thus obtaining $\boldsymbol{p}\,'$ and $p'_\mathrm{m} = \Gamma_{0,1,\ell'}(\boldsymbol{p}\,')$, and then mutating $\boldsymbol{b}$ with mutation rate p'_m; that is,

$$p_\mathrm{m} = \Gamma_{0,1,\ell'}(p_1 \ldots p_{\ell'}) \tag{C7.1.20}$$

$$p'_i = \begin{cases} p_i & u > p_\mathrm{m} \\ 1 - p_i & u \le p_\mathrm{m} \end{cases} \tag{C7.1.21}$$

$$p'_\mathrm{m} = \Gamma_{0,1,\ell'}(p'_1 \ldots p'_{\ell'}) \tag{C7.1.22}$$

$$b'_j = \begin{cases} b_j & u > p'_\mathrm{m} \\ 1 - b_j & u \le p'_\mathrm{m}. \end{cases} \tag{C7.1.23}$$

As usual, $u \sim U([0,1))$ denotes a uniform random variable sampled anew for each $i \in \{1, \ldots, \ell'\}$ and $j \in \{1, \ldots, \ell\}$.

This mutation mechanism was experimentally tested on a few continuous, high-dimensional test functions (the *sphere model*, the *weighted sphere model*, and the *generalized Rastrigin function*) with a genetic algorithm using (μ, λ) selection, and outperformed a canonical genetic algorithm with respect to convergence velocity as well as convergence reliability (Bäck 1992b). Concerning the selection method, these experiments demonstrated that the (μ, λ) selection (with $\mu = 10$, $\lambda = 50$) clearly outperformed proportional selection and facilitated much larger average mutation rates in the population than proportional selection did (for proportional selection, mutation rates quickly dropped to an average value of 0.001, roughly a value of $1/\ell$, while for $(10, 50)$ selection mutation rates as large as 0.005 were maintained by the algorithm). B2.7.4

Based on Bäck's work, Smith and Fogarty (1996) recently incorporated the self-adaptation method described by equations (C7.1.20)–(C7.1.23) into a *steady-state* (or $(\mu+1)$, in the terminology of evolution strategies) genetic algorithm, where just one new individual is created and substituted within the population at each cycle of the main loop of the algorithm. The new individual is generated by recombination (parents are chosen according to proportional selection), followed by an internal $(1, c)$ strategy which generates c mutants according to the self-adaptive method described above and selects one of them (either deterministically as usual, or according to proportional selection) as the offspring of the $(\mu+1)$ strategy. The authors investigated several policies for deleting an individual from the population (deletion of the worst, deletion of the oldest), recombination operators (which, however, had no clear impact on the results at all), mutation encoding (standard binary code, Gray code, exponential code), and the value of $c \in \{1, 2, 5, 10\}$ on *NK landscapes* with $N = 16$ and $K \in \{0, 4, 8, 15\}$. From these experiments, Smith and Fogarty derived a number of important conclusions regarding the best policies for the self-adaptation mechanism, namely: C2.7.3 B2.7.2

(i) Replacing the oldest of the population with the best offspring, conditional on the latter being the better of the two, is the best selection and deletion policy. Because replacing the oldest (rather than the worst) drops the elitist property of the $(\mu+1)$ strategy, this confirms observations from evolution strategies that self-adaptation needs a nonelitist selection strategy to work successfully (see above).
(ii) A value of $c = 5$ was consistently found to produce best results, such that the necessity to produce a surplus of offspring individuals as found by Bäck (1992b) and the $1/5$ success rule are both confirmed.
(iii) Gray coding and standard binary coding showed similar performance, both substantially outperforming the exponential encoding. On the most complex landscapes, however, the Gray coding also outperformed standard binary coding.

The comparison of the self-adaptive mutation mechanism with all standard fixed mutation rate settings (see Section E1.2 for an overview) clarified the general advantage of self-adaptation by significantly outperforming these fixed mutation rate settings. E1.2

The method described so far for self-adapting the mutation rates in canonical genetic algorithms was historically developed on the basis of the assumption that both the object variables and the strategy

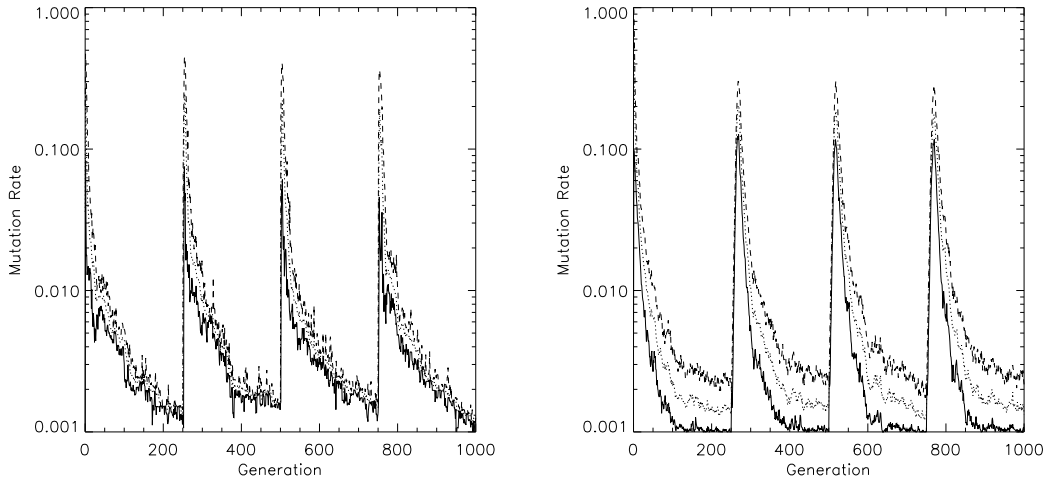


Figure C7.1.4. The self-adaptation of mutation rates is shown here for the time-varying counting ones function with $\ell = 1000$ and $(15, 100)$ selection, without recombination. The left-hand plot shows the minimum, average, and maximum mutation rates that occur in the population when the binary representation of p_m is used, while the right-hand plot shows the corresponding mutation rates when the mutation operator according to (C7.1.24) is used.

parameters should be represented by binary strings. It is clear from research on evolution strategies and evolutionary programming, however, that it should also be possible to incorporate the mutation rate $p_m \in [0, 1]$ directly into the genotype of individuals $\boldsymbol{a} = (\boldsymbol{b}, p_m) \in \{0, 1\}^\ell \times [0, 1]$ and to formulate a mutation operator that mutates p_m rather than its binary representation. Recently, Bäck and Schütz (1995, 1996) proposed a first version of such a self-adaptation mechanism, which was successfully tested for the mixed-integer problem of optimizing optical multilayer systems as well as for a number of combinatorial optimization problems with binary object variables.

Based on a number of requirements similar to those formulated by Schwefel for evolution strategies, namely that

(i) the expected change of p_m by repeated mutations should be equal to zero,
(ii) mutation of $p_m \in]0, 1[$ must yield a feasible mutation rate $p'_m \in]0, 1[$,
(iii) small changes should be more likely than large ones, and
(iv) the median should equal one,

these authors proposed a mutation operator of the form

$$p'_m = \left(1 + \frac{1 - p_m}{p_m} \exp(-\gamma N(0, 1))\right)^{-1} \tag{C7.1.24}$$

$$b'_j = \begin{cases} b_j & u > p'_m \\ 1 - b_j & u \leq p'_m \end{cases} \tag{C7.1.25}$$

where γ ($\gamma = 0.2$ was chosen in the experiments of the authors) is a learning rate analogue to τ_0 in equation (C7.1.4). A direct comparison of this operator with the one using a binary representation of p_m has not yet been performed, but it has already been observed by Bäck and Schütz (1996) that the learning rate γ is a critical parameter of equation (C7.1.24), because it determines the velocity of the self-adaptation process. In contrast to this, the method described by equations (C7.1.20)–(C7.1.23) eliminates the mutation rate completely as a parameter of the algorithm, such that it provides a more robust algorithm.

In analogy with the experiment performed with evolution strategies on the time-varying version of the sphere model, the self-adaptation mechanisms for mutation rates are tested with a time-varying version of the binary counting ones problem $f(\boldsymbol{b}) = \sum_{i=1}^{\ell} b_i \rightarrow \min$, which is modified by switching between f and $f'(\boldsymbol{b}) = \ell - f(\boldsymbol{b})$ every g generations. The experiment is performed for $\ell = 1000$ and $g = 250$ with a self-adaptive genetic algorithm using $(15, 100)$ selection, but without crossover. The results for the minimum, average, and maximum mutation rates are shown in figure C7.1.4 for the mutation mechanism

according to equations (C7.1.20)–(C7.1.23) (left-hand plot) and for the mutation mechanism according to equation (C7.1.24) (right-hand plot). It is clear from these figures that both mutation schemes facilitate the necessary adaptation of mutation rates, following a near-optimal schedule that exponentially decreases from large mutation rates ($p_m \approx 0.5$) at the beginning of the search to mutation rates of the order of $1/\ell$ in the final stage of the search. This behavior is in perfect agreement with the theoretical knowledge about the optimal mutation rate for the counting ones function (see e.g. Section E1.2, or the article by Bäck (1993)), but the available diversity of mutation rates in the population is smaller when the binary representation (left-hand plot) is used than with the continuous representation of p_m.

E1.2

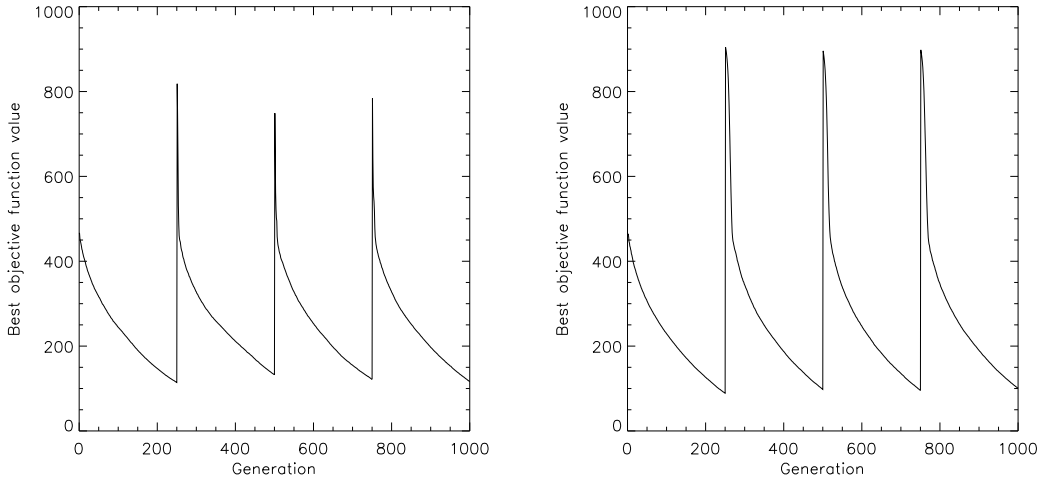


Figure C7.1.5. The best objective function values in the population for both self-adaptation mechanisms shown in figure C7.1.4.

The corresponding best objective function values in the population are shown in figure C7.1.5, and give clear evidence that the principle works well for both of the self-adaptation mechanisms presented here—thus again confirming the result of Beyer that the precise form of the probability density function used for modifying the mutation rates does not matter very much. Also in binary search spaces, self-adaptation is a powerful and robust technique, and the following sections demonstrate that this is true for other search spaces as well.

C7.1.2.3 Integer search spaces

For the general integer programming problem

$$\max\{f(\boldsymbol{x}) \mid \boldsymbol{x} \in M \subseteq \mathbb{Z}^n\} \tag{C7.1.26}$$

where $\mathbb{Z}$ denotes the set of integers, Rudolph (1994) presented an algorithm that self-adapts the total step size s of the variation of $\boldsymbol{x}$. By applying the principle of maximum entropy, i.e. the search for a distribution that is spread out as uniformly as possible without contradicting the given information, he demonstrated that a multivariate, symmetric extension of the geometric distribution is most suitable for the distribution of the variation $k = (k_1, \ldots, k_n)^T$ $(k_i \in \mathbb{Z})$ of the object variable vector $\boldsymbol{x}$. For a multivariate random variable Z, which is distributed according to the probability density function

$$P\{Z_1 = k_1, \ldots, Z_n = k_n\} = \prod_{i=1}^{n} P\{Z_i = k_i\} \tag{C7.1.27}$$

$$= \left(\frac{p}{2-p}\right)^n \prod_{i=1}^{n} (1-p)^{|k_i|} \tag{C7.1.28}$$

$$= \left(\frac{p}{2-p}\right)^n (1-p)^{\|k\|_1} \tag{C7.1.29}$$

where $\|k\|_1 = \sum_{i=1}^{n} |k_i|$ denotes the ℓ_1 norm, the expectation of the ℓ_1 norm of Z (i.e. the mean step size s) is given by

$$s = \mathbf{E}(\|Z\|_1) = n\frac{2(1-p)}{p(2-p)}. \tag{C7.1.30}$$

In full analogy with equation (C7.1.4), the mutation operator proposed by Rudolph modifies step size s and object variables x_i according to

$$s' = s\exp(\tau_0 N(0,1)) \tag{C7.1.31}$$

$$x_i' = x_i + G_i(p') \tag{C7.1.32}$$

where $G_i(p')$ denotes a realization of a one-dimensional random variable with probability density function

$$P\{G_i = k\} = \frac{p'}{2-p'}(1-p')^{|k|} \tag{C7.1.33}$$

i.e. the symmetric generalization of the geometric distribution with parameter p', and p' is obtained from s' according to equation (C7.1.30) as follows:

$$p' = 1 - \frac{s'/n}{1+(1+(s'/n)^2)^{1/2}}. \tag{C7.1.34}$$

Algorithmically, a realization $G_i(p')$ can be generated as the difference of two independent geometrically distributed random variables (both with parameter p'). A geometric random variable G is obtained from a uniform random variable U over $[0,1) \subset \mathbb{R}$ according to the transformation

$$G = \left\lfloor \frac{\log(1-U)}{\log(1-p')} \right\rfloor. \tag{C7.1.35}$$

Except for the integer representation and the mutation operator, the algorithm developed by Rudolph is based on an ordinary evolution strategy with (μ, λ) selection (with $\mu = 30$, $\lambda = 100$ for the experimental runs), intermediary recombination of the step sizes s, and no recombination of the object variable vector (i.e. when two individuals are recombined, their step sizes are averaged and the object variable vector is chosen from the first or second parent at random). The algorithm was empirically tested by Rudolph (1994) on five nonlinear integer programming problems and located the global optima of these problems for at least 80 percent of 1000 independent runs per problem.

C7.1.2.4 Finite-state machines

Methods to apply the self-adaptation principle to *finite-state machine* representations as used by evolutionary programming for sequence prediction tasks have recently been presented by Fogel *et al* (1994, 1995). These authors discuss two different methods for self-adaptation of mutability parameters (i.e. the probabilities of performing any of the possible *mutations* of state addition, state deletion, changes of output symbols, the initial state, and a next-state transition) associated with each component of a finite-state machine. The mutability parameters p_i are all initially set to a minimum value of 0.001 and mutated according to a historically older version of equation (C7.1.16) (Fogel *et al* 1991):

C1.5
C3.2.4

$$p_i' = p_i + \alpha N(0,1) \tag{C7.1.36}$$

where $\alpha = 0.01$.

The methods are different with respect to the selection of a machine component for mutation. In *selective self-adaptation*, a component is selected for each type of mutation on the basis of relative selection probabilities $p_i / \sum p_k$, where p_i is the mutability parameter for the ith component and the summation is taken with k running over all components. For this scheme, separate mutability parameters are maintained for each state, each output symbol, and each next-state transition.

In contrast to this, *multimutational self-adaptation* denotes a mechanism where each mutability parameter designates the absolute probability of modification for that particular component, such that the probability for each component to be mutated is independent of the probabilities of other components. Consequently, multimutational self-adaptation is expected to offer greater diversity in the types of offspring

machine that could be generated from a parent, and the learning rate α of the approach is extremely important under this approach ($0.005 \leq \alpha \leq 0.999$ was generally enforced in the experiments).

Both approaches were tested by the authors on two simple prediction tasks and consistently outperformed the evolutionary programming method without self-adaptation. While on the simpler problem the selective self-adaptation method had a slightly better performance than the multimutational method, the latter performed better on the more complex problem, thus indicating the need to explore a larger diversity by the mutation operator in this case. Certainly, more work is needed to assess the strengths and weaknesses of both approaches for self-adaptation on finite-state machines, but once again the robustness and wide applicability of this general principle for on-line learning of mutational control parameters has been demonstrated by these experiments.

C7.1.3 Recombination operators

In contrast to the mutation operator, recombination in evolutionary algorithms has received much less attention for self-adaptation of the operator's characteristics (e.g. the number and location of crossover points) and parameters (e.g. the application probability or segment exchange probability). This can be explained in part by the historical emergence of the algorithmic principle in connection with the mutation operator in evolution strategies and evolutionary programming, but it might also be argued that the implicit link between strategy parameters and their impact on the fitness of an individual is not as tight for recombination as the self-adaptation idea requires (this argument is supported by recent empirical findings indicating that the set of problems where crossover is useful might be smaller than expected (Eshelman and Schaffer 1993)). Research concerning the self-adaptation of recombination operators is still in its infancy, and the two examples reported here for binary search spaces can only give an impression of the many open questions that need to be answered in the future.

C7.1.3.1 Binary search spaces

For canonical genetic algorithms and a *multipoint crossover* operator, Schaffer and Morishima (1987) C3.3.1 developed a method for self-adaptation of both the number and location of crossover points. These strategy parameters (i.e. the crossover points) are incorporated in an individual by attaching to the end of each object variable vector $\boldsymbol{b} = (b_1 \ldots b_\ell) \in \{0, 1\}^\ell$ another binary vector $\boldsymbol{c}$ of the same length. The bits of this strategy parameter vector are interpreted as *crossover punctuations*, where a one at position i in vector $\boldsymbol{c}$ indicates that a crossover point occurs at the corresponding position i in the object variable vector. During initialization, the probability of generating a one in the vector $\boldsymbol{c}$ is set to a rather small value $p_{\mathrm{xo}} = 0.04$.

Crossover between two strings then works by copying the bits from each parent string one by one to one of the offspring from left to right, and when a punctuation mark is encountered in either parent, the bits begin going to the other offspring. The punctuation marks themselves are also passed on to the offspring, when this happens, such that the strategy parameter vector is also subject to recombination. Furthermore, the usual mutation by bit inversion is also applied to the strategy parameter vector, such that the principles of self-adaptation are fully applied in the method proposed by Schaffer and Morishima (1987).

For an empirical investigation of the *punctuated crossover* method, the authors used the five *test functions* proposed by De Jong (1975) and the on-line performance measure (i.e. an average of all trials in a run). B2.7.4 The self-adaptive crossover operator was found to statistically outperform a canonical genetic algorithm on four of the five functions while being no worse on the other. Looking at the dynamics of the distribution of punctuation marks over time, the authors observed that some loci tend to accumulate more punctuation marks than others as time progresses and the locations of these concentrations change over time. Concerning the dynamics of the average number of crossover events that occur per mating, the authors found that the number of 'productive' crossover events (i.e. those events where nonidentical gene segments are swapped between parents and offspring different from the parents is produced) remained nearly constant and correlated strongly with ℓ (and implicitly with n, the dimension of the real-valued search space the binary strings are mapped to in this parameter optimization task). These results are certainly interesting and deserve a more detailed investigation, especially under experimental conditions where the effect of a recombination operator is well understood and the optimal operator to be encountered by self-adaptation is known in advance. The *sphere model* with continuous representation of variables could B2.7.4

be a good candidate for such an experiment, because the theory of discrete and intermediary recombination is relatively well understood for this function (Beyer 1995a).

As clarified by Spears (1995), it is not clear whether the success of Schaffer and Morishima's punctuated crossover is due to the self-adaptation of crossover points or whether it stems from the simple fact that they compared a canonical genetic algorithm with one-point crossover with the self-adaptive method that, on average, was using n-point crossover with $n > 1$. Spears (1995) investigated a simple method called one-bit self-adaptation, where a single strategy parameter bit added to an individual indicated whether uniform crossover or two-point crossover was performed on the parents (ties are broken randomly). Experiments were performed on the so-called *N-peak problems*, in which each problem has one optimal solution and $N - 1$ suboptimal solutions. The cases $N \in \{1, 6\}$ were investigated for 30 bits and 900 bits, where the latter problem contains 870 dummy bits that do not change the objective function, but have some impact on the effectiveness of the crossover operators.

The author performed a control experiment to determine the best crossover operator on these problems, and then ran the self-adaptation method on the problems. Concerning the best-so-far curves, the performance of self-adaptation consistently approached the performance of the best crossover operator alone, but in the case of the six-peak 900-bit problem, the dominant operator chosen by self-adaptation (two-point crossover in more than 90% of all cases) was *not* the operator identified in the control experiment (uniform crossover). As an overall conclusion of further experiments along this line, Spears (1995) clarified that the key feature of the 'self-adaptation' method used here is not to provide the possibility of adapting towards choosing the best operator, but rather the diversity provided to the algorithm by giving it access to two different recombination operators for exploitation during the search. Consequently, it might be worthwhile to run an evolutionary algorithm with a larger set of search operators than is customary, even if the algorithm is not self-adaptive, and the aspect of an additional benefit caused by the available diversity of strategy parameters is emphasized again by this observation.

C7.1.4 Conclusions

Numerous approaches for the self-adaptation of strategy parameters within evolutionary algorithms as discussed in the previous sections clarify that the fundamental underlying principle of self-adaptation—evolving both the object variables *and* the strategy parameters simultaneously—works under a variety of conditions regarding the search space and the variation mechanism for strategy parameters by exploiting the implicit link between strategy parameters (or *internal models*) of the individuals and their fitness. This seems to work best for the unary mutation operator, because the strategy parameters have a direct impact on the object variables of a single individual and are either selected for survival and reproduction or discarded, depending on whether their impact is beneficial or detrimental.

This robustness of the method clearly indicates that a fundamental principle of evolutionary processes is utilized here, and in fact it is worth mentioning that the base pair mutation rate of mammalian organisms is in part regulated by its own genotype by *repair enzymes* and *mutator genes* encoded on the DNA. The former are able to repair a variety of damage of the genome, while the latter increase the mutation rate of other parts of the genome (see Gottschalk 1989, pp 269–71, 182).

Concerning its relevance as a parameter control mechanism in evolutionary algorithms, self-adaptation clearly has the advantage of reducing the number of exogenous control parameters of these algorithms and thereby releasing the user from the costly process of fine tuning these parameters by hand. Moreover, it is well known that constant control parameter settings (e.g. of the mutation rate in canonical genetic algorithms) are far from being optimal and that self-adaptation principles are able to generate a nearly optimal dynamic schedule of these parameters (see e.g. the article by Bäck (1992a) and the examples for the time-varying sphere model and counting ones function as presented in section C7.1.2).

While the principle is of much practical usefulness and has demonstrated its power and robustness in many examples, many open questions remain to be answered. The most important questions are those for the optimal conditions for self-adaptation concerning the choice of a selection operator, population sizes, and the probability density function used for strategy parameter modifications, i.e. for the algorithmic circumstances required. This also raises the question for an appropriate optimality criterion for self-adaptation, having in mind that maximizing speed of adaptation might be good for holding an optimum within dynamically changing environments rather than for emphasizing global convergence properties. The speed of adaptation is controlled in some of the self-adaptation approaches by exogenous learning rates (e.g. τ_0 in equation (C7.1.4), τ and τ' in equation (C7.1.6), and α in equation (C7.1.16)), and the 'optimal'

setting of these learning rates usually emphasizes convergence velocity rather than global convergence of the evolutionary algorithm, such that for multimodal objective functions a different setting of the learning rates, implying slower adaptation, might be more appropriate. Finally, it is important to recognize that the term self-adaptation is used to characterize a wide spectrum of different possible behaviors, ranging from the precise learning of the time-dependent optimal setting of a single control parameter (such as σ for the time-varying sphere model) to the creation of a diversity of different strategy parameter values which are available in the population for utilization by the individuals.

It has always been emphasized by Schwefel (1987, 1989, 1992) that diversity of the internal models is a key ingredient to the synergistic effect of self-adaptation, facilitating a collection of individuals equipped with imperfect, diverse internal models of their environment to perform collectively as well as or even better than a single expert individual with precise, full knowledge of the environment does. While some of the implementations of self-adaptation certainly exploit more the diversity of parameter settings rather than adapting them, the key to the success of self-adaptation seems to consist in using both sides of the coin to facilitate reasonably fast adaptation (and, as a consequence, a good convergence velocity) and reasonably large diversity (and a good convergence reliability) at the same time.

References

Angeline P J 1996 The effects of noise on self-adaptive evolutionary optimization *Proc. 5th Ann. Conf. on Evolutionary Programming* ed L J Fogel, P J Angeline and T Bäck (Cambridge, MA: MIT Press)

Arabas J, Michalewicz Z and Mulawka J 1994 GAVaPS—a genetic algorithm with varying population size *Proc. 1st IEEE Conf. on Evolutionary Computation (Orlando, FL, June 1994)* (Piscataway, NJ: IEEE) pp 73–8

Bäck T 1992a The interaction of mutation rate, selection, and self-adaptation within a genetic algorithm *Parallel Problem Solving from Nature, 2 (Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature, Brussels, 1992)* ed R Männer and B Manderick (Amsterdam: Elsevier) pp 85–94

——1992b Self-adaptation in genetic algorithms *Proc. 1st Eur. Conf. on Artificial Life* ed F J Varela and P Bourgine (Cambridge, MA: MIT Press) pp 263–71

——1993 Optimal mutation rates in genetic search *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 2–8

Bäck T and Schütz M 1995 Evolution strategies for mixed-integer optimization of optical multilayer systems *Proc. 4th Ann. Conf. on Evolutionary Programming (San Diego, CA, March 1995)* ed J R McDonnell, R G Reynolds and D B Fogel (Cambridge, MA: MIT Press) pp 33–51

——1996 Intelligent mutation rate control in canonical genetic algorithms *Foundations of Intelligent Systems, 9th Int. Symp., ISMIS '96 (Lecture Notes in Artificial Intelligence 1079)* ed Z W Ras and M Michalewicz (Berlin: Springer) pp 158–67

Bäck T and Schwefel H-P 1996 Evolutionary computation: an overview *Proc. 3rd IEEE Conf. on Evolutionary Computation* (Piscataway, NJ: IEEE) pp 20–9

Beyer H-G 1995a Toward a theory of evolution strategies: on the benefits of sex—the $(\mu/\mu, \lambda)$-theory *Evolutionary Comput.* **3** 81–111

——1995b Toward a theory of evolution strategies: self-adaptation *Evolutionary Comput.* **3** 311–48

Davis L 1989 Adapting operator probabilities in genetic algorithms *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, June 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 61–9

Davis M W 1994 The natural formation of Gaussian mutation strategies in evolutionary programming *Proc. 3rd Ann. Conf. on Evolutionary Programming (San Diego, CA, February 1994)* ed A V Sebald and L J Fogel (Singapore: World Scientific) pp 242–52

De Jong K A 1975 *An Analysis of the Behaviour of a Class of Genetic Adaptive Systems* PhD Thesis, University of Michigan

Eiben A E and Michalewicz Z 1996 Personal communication

Eshelman L J and Schaffer J D 1993 Crossover's niche *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 9–14

Fogel D B 1992 *Evolving Artificial Intelligence* PhD Thesis, University of California

Fogel D B, Fogel L J and Atmar W 1991 Meta-evolutionary programming *Proc. 25th Asilomar Conf. on Signals, Systems and Computers (Pacific Grove, CA)* ed R R Chen, pp 540–5

Fogel L J, Angeline P J and Fogel D B 1995 An evolutionary programming approach to self-adaptation on finite state machines *Proc. 4th Ann. Conf. on Evolutionary Programming (San Diego, CA, March 1995)* ed J R McDonnell, R G Reynolds and D B Fogel (Cambridge, MA: MIT Press) pp 355–65

Fogel L J, Fogel D B and Angeline P J 1994 A preliminary investigation on extending evolutionary programming to include self-adaptation on finite state machines *Informatica* **18** 387–98

Gehlhaar D K and Fogel D B 1996 Tuning evolutionary programming for conformationally flexible molecular docking *Proc. 5th Ann. Conf. on Evolutionary Programming* ed L J Fogel, P J Angeline and T Bäck (Cambridge, MA: MIT Press)

Gottschalk W 1989 *Allgemeine Genetik* (Stuttgart: Thieme)

Rechenberg I 1994 *Evolutionsstrategie '94 (Werkstatt Bionik und Evolutionstechnik 1)* (Stuttgart: Frommann–Holzboog)

Rudolph G 1992 On correlated mutations in evolution strategies *Parallel Problem Solving from Nature, 2 (Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature, Brussels, 1992)* ed R Männer and B Manderick (Amsterdam: Elsevier) pp 105–14

——1994 An evolutionary algorithm for integer programming *Parallel Problem Solving from Nature—PPSN III (Proc. Int. Conf. on Evolutionary Computation and 3rd Conf. on Parallel Problem Solving from Nature, Jerusalem, October 1994)* (*Lecture Notes in Computer Science 866*) ed Yu Davidor, H-P Schwefel and R Männer (Berlin: Springer) pp 139–48

Saravanan N 1994 Learning of strategy parameters in evolutionary programming: an empirical study *Proc. 3rd Ann. Conf. on Evolutionary Programming (San Diego, CA, February 1994)* ed A V Sebald and L J Fogel (Singapore: World Scientific)

Saravanan N and Fogel D B 1994 Evolving neurocontrollers using evolutionary programming *Proc. 1st IEEE Int. Conf. on Evolutionary Computation (Orlando, FL, June 1994)* vol 1 (Piscataway, NJ: IEEE) pp 217–22

Saravanan N, Fogel D B and Nelson K M 1995 A comparison of methods for self-adaptation in evolutionary algorithms *BioSystems* **36** 157–66

Schaffer J D and Morishima A 1987 An adaptive crossover distribution mechanism for genetic algorithms *Proc. 2nd Int. Conf. on Genetic Algorithms (Cambridge, MA, July 1987)* ed J J Grefenstette (Hillsdale, NJ: Erlbaum) pp 36–40

Schlierkamp-Voosen D and Mühlenbein H 1996 Adaptation of population sizes by competing subpopulations *Proc. 3rd IEEE Conf. on Evolutionary Computation* (Piscataway, NJ: IEEE) pp 330–5

Schwefel H-P 1977 *Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie (Interdisciplinary Systems Research 26)* (Basel: Birkhäuser)

——1987 Collective intelligence in evolving systems *Ecodynamics, Contributions to Theoretical Ecology* ed W Wolff, C-J Soeder and F R Drepper (Berlin: Springer) pp 95–100

——1989 Simulation evolutionärer Lernprozesse *Erwin–Riesch Workshop on Systems Analysis of Biomedical Processes, Proc. 3rd Ebernburger Working Conf. of ASIM/GI* ed D P F Möller (Braunschweig: Vieweg) pp 17–30

——1992 Imitating evolution: collective, two-level learning processes *Explaining Process and Change—Approaches to Evolutionary Economics* ed U Witt (Ann Arbor, MI: University of Michigan Press) pp 49–63

Smith J and Fogarty T C 1996 Self adaptation of mutation rates in a steady state genetic algorithm *Proc. 3rd IEEE Conf. on Evolutionary Computation* (Piscataway, NJ: IEEE) pp 318–23

Spears W M 1995 Adapting crossover in evolutionary algorithms *Proc. 4th Ann. Conf. on Evolutionary Programming (San Diego, CA, March 1995)* ed J R McDonnell, R G Reynolds and D B Fogel (Cambridge, MA: MIT Press) pp 367–84

Yao X and Liu Y 1996 Fast evolutionary programming *Proc. 5th Ann. Conf. on Evolutionary Programming* ed L J Fogel, P J Angeline and T Bäck (Cambridge, MA: MIT Press)

C7.2 Metaevolutionary approaches

Bernd Freisleben

Abstract

This section presents approaches to determining the optimal evolutionary algorithm (EA), that is, the best types of evolutionary operator and their parameter settings for a given problem. The basic idea is to consider the search for the best EA as an optimization problem and use another EA to solve it. In such a metaevolutionary approach, a metalevel EA operates on a population of base-level EAs which in turn solve the problem in discussion. The section presents an informal and a formal description of the general metaevolutionary approach, provides pseudocode for realizing it, discusses some theoretical results, and reviews related work published in the literature.

C7.2.1 Working mechanism

After having defined the individuals of a population for a given problem, the designer of an evolutionary algorithm (EA)† is faced with the problem of deciding what types of operator and control parameter settings are likely to produce the best results. For example, the decisions to be made might include choices among the different variants of the *selection*, *crossover*, and *mutation* operators which have been suggested in the literature (Bäck 1996, Goldberg 1989a, Michalewicz 1994), and, depending on the operators, values for the corresponding control parameters such as the crossover probability, the mutation probability, and the population size (Booker 1987, De Jong 1975, Goldberg 1989b, Hesser and Männer 1990, Mühlenbein and Schlierkamp-Voosen 1995, Schaffer *et al* 1989) must be determined. The decision may be based on:

C2, C3.3, C3.2

- systematically checking a range of operators and/or parameter values and assessing the performance of the EA (De Jong 1975, Schaffer *et al* 1989)
- the experiences reported in the literature describing similar application scenarios (Goldberg 1989a, Jog *et al* 1989, Oliver *et al* 1987, Starkweather *et al* 1991)
- the results of theoretical analyses for determining the optimal parameter settings (Goldberg 1989b, Hesser and Männer 1990, Nakano *et al* 1994).

Since these proposals are typically not universally valid, and therefore it may be that none of them produces satisfactory results for the particular problem considered, a more promising approach is to consider the search for the 'best' EA for a given problem as an optimization problem itself. This *metaoptimization* problem can then be solved by any of the general purpose optimization methods proposed in the literature, including well-known heuristic methods (Reeves 1993) such as *simulated annealing* (van Laarhoven and Aarts 1987) or *tabu search* (Glover 1990)—but also by any type of evolutionary algorithm, leading to a *metaevolutionary* approach.

D3.5.2

D3.5.4

In order to realize a metaevolutionary approach, a two-level optimization strategy is required, as shown in figure C7.2.1. At the top level, a *metalevel* EA operates on a population of *base-level* EAs, each of which is represented by a separate individual. At the bottom level, the base-level EAs work on a population of individuals which represent possible solutions of the problem to be solved. Each of the base-level EAs runs independently to produce a solution of the problem considered, and the fitness of the

† The term *evolutionary algorithm* is used to denote any method of evolutionary computation, such as genetic algorithms (GA), evolution strategies (ESs), evolutionary programming (EP), and genetic programming (GP).

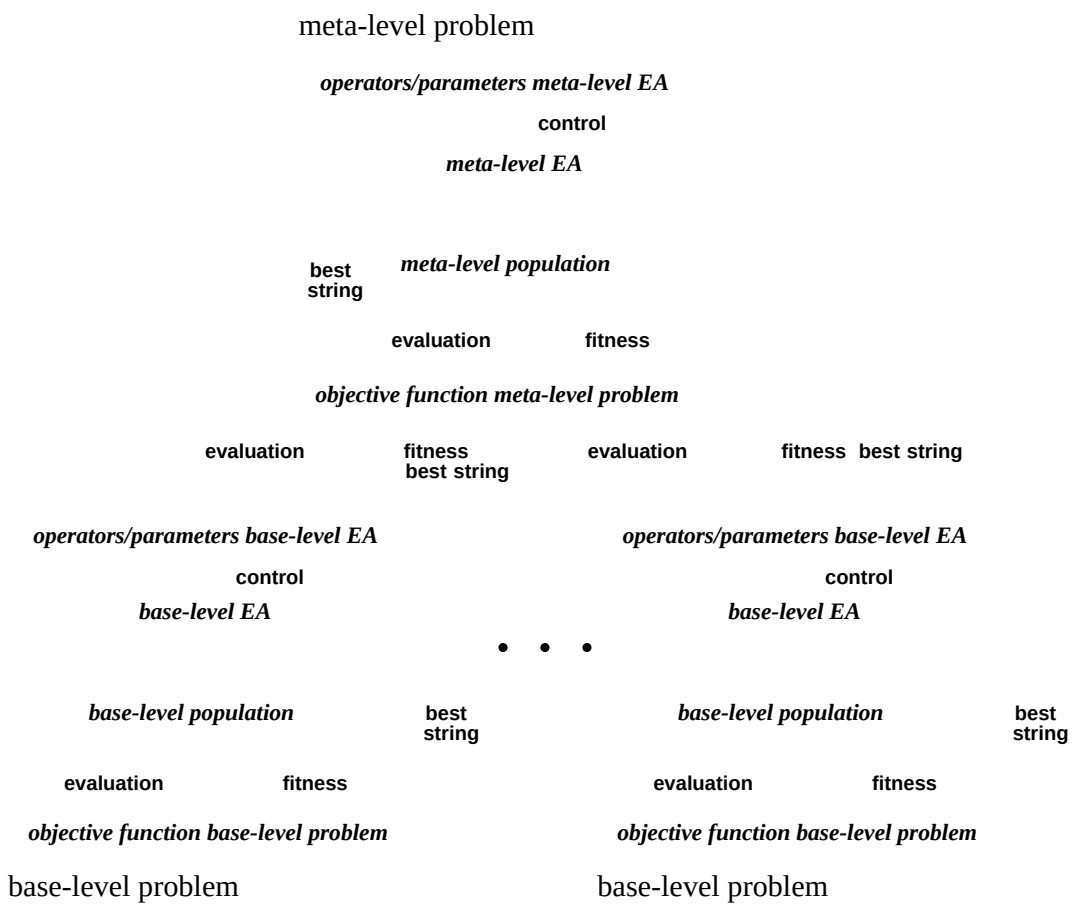


Figure C7.2.1. A metaevolutionary approach.

solution influences the operation of the metalevel EA; the fitness of an individual representing a base-level EA is taken to be the fitness of the best solution found by the base-level EA in the entire run using the current parameters. The numbers of generations created on the two levels are independent of each other. The individual with the highest fitness ever found is expected to be the best EA for the original problem.

The information that needs to be represented to encode a base-level EA as a member of the metalevel population depends on the nature of the base-level problem. In general, since an EA is characterized by its operators and the parameter values to control them, there are essentially two kinds of information that need to be represented in the individuals of the metalevel population: first the particular variant of an evolutionary operator, and second the parameter values required for the selected variant. In some sense, this is a kind of variable-dimensional structure optimization problem.

The above description is implicitly based on the classical (steady-state) model of evolutionary computation, in which the operators and operator probabilities are specified before the start of an EA and remain unchanged during its run. However, several proposals have been made to allow the adaptation of the operator probabilities as the run progresses, e.g. in order to facilitate schema identification and reduce schema disruption (Davis 1991, Ostermeier *et al* 1994, White and Oppacher 1994). Furthermore, there is some empirical evidence that the most effective operator variants also vary during the search process (Davis 1991, Michalewicz 1994). Since in a metaevolutionary approach the overall behavior of a base-level EA is typically used to evaluate its fitness in the metalevel population, the concept of *self-adaptation* (i.e. the evolution of strategy parameters on-line during the search, see Bäck 1996) is more suitable to support dynamically adaptive models of evolutionary computation. However, an advanced version of a metaevolutionary approach might nevertheless be beneficial to analyze the impact of particular operator variants and parameter values on the solution quality during the different stages of the evolution process (Freisleben and Härtfelder 1993a, b) and might therefore provide valuable hints for dynamical operator/parameter adaptation.

C7.1

C7.2.2 Formal description

Let B be a base-level problem, I_B its solution space, $\boldsymbol{x} = (x_1, \ldots, x_m) \in I_B$ a member of the solution space, and $F_B : I_B \rightarrow \mathbb{R}$ the objective function to be optimized (without loss of generality, we assume that F_B should be maximized, i.e. we are looking for an $\boldsymbol{x}^*$ such that $F_B(\boldsymbol{x}^*) \geq F_B(\boldsymbol{x}) \quad \forall \boldsymbol{x} \in I_B$).

Furthermore, let $EA_{[V,F,I]} : V \rightarrow I$ be a generic evolutionary algorithm parameterized by the space V of vectors representing all possible settings of operator variants and parameters, a fitness function F, and the space I of individuals, which for each vector $\boldsymbol{v} \in V$ returns an individual $\boldsymbol{x}_v = EA_{[V,F,I]}(\boldsymbol{v}) \in I$ with the best possible fitness.

Based on this generic definition, let us consider an evolutionary algorithm $EA_{[V_B,F_B,I_B]} : V_B \rightarrow I_B$ for problem B. In order to find a good solution of problem B, our goal is to find a parameter setting $\boldsymbol{v}_B^* \in V_B$ such that

$$F_B(EA_{[V_B,F_B,I_B]}(\boldsymbol{v}_B^*) \geq F_B(EA_{[V_B,F_B,I_B]}(\boldsymbol{v}_B) \qquad \forall \boldsymbol{v}_B \in V_B. \tag{C7.2.1}$$

Thus, the search for a good solution for the base-level problem B reduces to a metalevel problem M of maximizing the objective function $F_M : V_B \rightarrow \mathbb{R}$,

$$F_M(\boldsymbol{v}_B) = F_B(EA_{[V_B,F_B,I_B]}(\boldsymbol{v}_B)). \tag{C7.2.2}$$

The definition of the objective function (C7.2.2) allows us to treat the search for the best evolutionary algorithm for problem B as an optimization problem which may be solved by a metaevolutionary algorithm $EA_{[V_M,F_M,V_B]} : V_M \rightarrow V_B$, where V_M is the space of all possible operator and parameter settings for the metalevel problem M. The aim of the metaevolutionary algorithm is to find optimal parameter values

$$\boldsymbol{v}_B^* = EA_{[V_M,F_M,V_B]}(\boldsymbol{v}_M) \tag{C7.2.3}$$

where $\boldsymbol{v}_M$ is the parameter setting useed for the metaevolutionary algorithm.

C7.2.3 Pseudocode

The pseudocode for an EA which is used to realize a metaevolutionary approach is presented below. The code is based on the selection, recombination, mutation and replacement sequence of operations typically found in the genetic algorithm paradigm (see Sections B1.1 and B1.2 for a more detailed description); in this particular version, replacement of old individuals by new ones (line 8) is performed outside the **for**-loop (line 5); that is, the new population is created after all offsprings of a generation have been produced. Note that in order to distinguish between the base- and the metalevel EA, two new parameters (F, I) have been introduced in addition to the ones given in the pseudocode presented in Sections B1.1 and B1.2.

B1.1, B1.2

B1.1, B1.2

Input: $\mu, \lambda, t_{max}, F, I$
Output: $\boldsymbol{x}^* \in I$, the best individual ever found.
procedure EA;
1 $t \leftarrow 0$;
2 $P(t) \leftarrow$ initialize(μ, I);
3 $\boldsymbol{F}(t) \leftarrow$ evaluate($P(t), \mu$);
4 **while** ($\iota(P(t), \boldsymbol{F}(t), t_{max}) \neq$ true) **do**
5 **for** $i \leftarrow 1$ **to** λ **do**
$P'(t) \leftarrow$ select($P(t), \boldsymbol{F}(t)$);
$\boldsymbol{a}_i'(t+1) \leftarrow$ recombine($P'(t)$);
$\boldsymbol{a}_i''(t+1) \leftarrow$ mutate($\boldsymbol{a}_i'(t+1)$);
od
6 $P'(t+1) \leftarrow (\boldsymbol{a}_1''(t+1), \ldots, \boldsymbol{a}_\lambda''(t+1))$;
7 $\boldsymbol{F}'(t+1) \leftarrow$ evaluate($P'(t+1)$), λ);
8 $\{P(t+1), \boldsymbol{F}(t+1)\} \leftarrow$ replace($P(t), P'(t+1), \boldsymbol{F}(t), \boldsymbol{F}'(t+1)$);
9 $t \leftarrow t+1$;
od

In order to implement a metaevolutionary approach, the procedure *EA* is called as follows:

$$\text{Best\_EA} \leftarrow \text{EA}(\mu_{\text{M}}, \lambda_{\text{M}}, t_{\text{max\_M}}, F_{\text{M}}, I_{\text{M}}).$$

The result of this call, Best_EA, is the best EA ever found for the given base-level problem. The first three parameters of the procedure, μ_{M}, λ_{M}, and $t_{\text{max\_M}}$, denote the population size, offspring population size, and maximum number of generations for the metalevel EA, respectively. F_{M} is the fitness function for the metalevel problem, and I_{M} is the space of individuals $\boldsymbol{a}_i(t)$ representing EA operator/parameter settings for the base-level problem.

The metaevolutionary approach is realized by defining F_{M} to include a call to procedure *EA* as follows:

$$F_{\text{M}}(\boldsymbol{a}_i(t)) = F_{\text{B}}(\text{EA}(\pi_{i_\mu}(\boldsymbol{a}_i(t)), \pi_{i_\lambda}(\boldsymbol{a}_i(t)), \pi_{i_{t_{\max}}}(\boldsymbol{a}_i(t)), F_{\text{B}}, I_{\text{B}})) \qquad \forall \boldsymbol{a}_i(t) \in I_{\text{M}}$$

where:

- $\pi_{i_\mu}(\boldsymbol{a}_i(t))$, $\pi_{i_\lambda}(\boldsymbol{a}_i(t))$, and $\pi_{i_{t_{\max}}}(\boldsymbol{a}_i(t))$ denote the parent population size, offspring population size, and maximum number of generations for the base-level problem, respectively; they are obtained by applying the projection operator π_i to the current metalevel individual $\boldsymbol{a}_i(t)$.
- F_{B} is the fitness function for the base-level problem.
- I_{B} is the space of individuals representing solutions to the base-level problem.

C7.2.4 Parameter settings

In a metaevolutionary approach, it is natural to ask how the types of operator and their parameter settings for the metalevel EA are determined. Several possibilities are described below.

(i) The operators and the parameter values of the metalevel EA are specified by the designer before the EA starts, and they do not change during the run. This approach is used by Bäck (1994), Grefenstette (1986), Lee and Takagi (1994), Mercer and Sampson (1978), Pham (1994), Shahookar and Mazumder (1990).

(ii) The operators and parameter values are initially determined by the designer and the metalevel EA is applied to find the best EA for a base-level problem which closely resembles the properties of the metalevel problem. Such a problem would require, for example, real-valued genes, the possibility to treat logical subgroups of genes as an atomic unit, and a sufficiently complex search space with multiple suboptimal peaks. The values obtained for the best base-level EA in this scenario are then copied and used as the parameter settings of the metalevel EA for optimizing the base-level EAs for the particular problem to be solved. This approach is used by Freisleben and Härtfelder (1993a).

(iii) The operators and the parameter values are initially determined by the designer and then copied from the best base-level EA for the problem in discussion (possibly after every metalevel generation). This, however, is only possible if the structural properties of the two problem types are identical. For example, it does not work if the base-level EA operates on strings for permutation problems, such as in combinatorial optimization problems like the *traveling salesman problem* (Freisleben and Merz 1996a, b, Goldberg and Lingle 1985, Oliver *et al* 1987). This approach is used by Freisleben and Härtfelder (1993b). G9.5

C7.2.5 Theory

Some theoretical results which may assist in finding useful parameter settings for EAs have been reported in the literature. For example, there are theoretical investigations of the optimal *population sizes* (Goldberg 1989b, Nakano *et al* 1994, Ros 1989), the optimal mutation probabilities (Bäck 1993, De Jong 1975, Hesser and Männer 1990), the optimal crossover probabilities (Schaffer *et al* 1989), and the relationships between several evolutionary operators (Mühlenbein and Schlierkamp-Voosen 1995) with respect to simple function optimization problems. E1.1

However, it was shown by Hart and Belew (1991) that a universally valid optimal parametrization of a GA does not exist, because the optimal parameter values are strongly dependent on the particular optimization problem to be solved by the GA, its representation, the population model, and the operators used. The large number of possibilities precludes an exhaustive search of the space of operators and operator probabilities. This is a strong argument in favor of a (heuristic) metaevolutionary or a *self-adaptive* approach. C7.1

C7.2.6 Related work

Several metaevolutionary approaches have been proposed in the literature, as discussed below.

(i) Mercer and Sampson (1978) gave presumably one of the first descriptions of a metaevolutionary approach. The authors investigated the effects of two crossover and three mutation operators on a simple pattern recognition problem; the aim of their meta-algorithm was to determine the most suitable operator probabilities. They started with a population size of five individuals in the metalevel population and performed 75 generations, and their results were based on a single run (probably due to the limited computing power available at that time). Mercer and Sampson defined special types of metaoperator which were different from the genetic operators used in the base-level EAs. The parameter values of the base-level algorithm were adaptable over time: different operator probabilities were used in the different stages of the search.

(ii) Grefenstette's meta-GA (Grefenstette 1986) operated on individuals representing the population size, crossover probability, mutation rate, generation gap, scaling window, and selection strategy (i.e. proportional selection in *pure* and *elitist* form). The six-dimensional parameter space was discretized, and only a small number (8 or 16) of different values was permitted for each parameter. The base-level problems investigated were De Jong's set of test functions (De Jong 1975), and the control parameters of the meta-GA were simply set to those identified by De Jong in a number of experiments (population size $\lambda = 50$, crossover probability $p_c = 0.6$, mutation probability $p_m = 0.001$, a generation gap of 1.0, a scaling window of seven, elitist selection) (De Jong 1975). The meta-GA started with a population size of 50 and produced 20 generations. The results suggested the use of a significantly different crossover probability and a different selection scheme for the on-line and off-line performance of the GAs, respectively. In particular, the meta-evolution experiment yielded the result $\lambda = 30$ (80), $p_c = 0.95$ (0.45), $p_m = 0.01$ (0.01), a generation gap of 1.0 (0.9), a scaling window of one (one), elitist (pure) selection when the on-line (off-line) performance was used. C2.2, C2.7

(iii) Shahookar and Mazumder (1990) used a meta-GA to optimize the crossover rate (for three different permutation crossover operators), the mutation rate, and the inversion rate of a GA to solve the standard cell placement problem for industrial circuits consisting of between 100 and 800 cells. Similar to Grefenstette's proposal, the individuals representing the GAs consisted of discrete (integer) values in a limited range. For the meta-GA, the population size was 20, the crossover probability 1.0, the mutation probability 0.2, and it was run for 100 generations. In the experiments it was observed that the GA parameter settings produced by the meta-GA approach need to examine 19–50 times fewer configurations as compared to a commercial cell placement system, and the runtimes were comparable.

(iv) Freisleben and Härtfelder (1993a, b) proposed a meta-GA approach which was based on a much larger space of up to 20 components, divided into *decisions* and *parameters*. A decision is numerically represented by the probability that a particular variant of an operator is selected among a limited number of variants available of that operator, and a parameter value is encoded as a real number associated with the selected variant. The authors performed two experiments, one in which the base-level GAs tried to solve a *neural network weight optimization problem* (strings with real-valued alleles) (Freisleben and Härtfelder 1993b) and another one where a set of differently complex symmetric *traveling salesman problems* (strings with integer-valued alleles) (Freisleben and Härtfelder 1993a) was solved. By providing means for investigating the significance of a decision for or against an operator variant on the solution process, and the significance of finding the optimal parameter value for a particular given operator variant, the authors demonstrated that making the right choice for some operator variants is more crucial than for others, and thus some light was shed on the relationships between the various GA features during the search process. For example, in the neural network weight optimization problem the operators which contributed most to the solution were the *mutation value replacement method* (adding a value to the old value to obtain the mutated value or overwriting the old value), the granularity of the *crossover operator* (applying crossover such that logical subgroups of genes stay together as a structural unit), the *selection method*, the distribution of the *mutation function*, and the decision for or against using the *elitist model*. The decision to apply the *crowding method* and the decision for mutating logical subgroups (i.e. the weights of a neuron) together became important after the 60th metalevel generation; that is, when other decisions and parameters had D1.2, G9.5, C3.2, C3.3, C2, C3, C2.7

already been appropriately determined. In the experiment with the set of traveling salesman problems it was surprising to find that the type of permutation *crossover operator*) was far less significant than the choice of the mutation method or the use of the elitist model. C3.3

Furthermore, in this approach the parameter values of the base-level GAs are adaptable to the complexity of the problem instances. The results were based on 100 metalevel generations with a starting population size of 50. On the level of the meta-GA, the authors adopted the currently best base-level parameter settings dynamically after every metalevel generation (Freisleben and Härtfelder 1993b), and also used the parameter settings obtained in a previous meta-GA experiment (Freisleben and Härtfelder 1993a).

(vi) Bäck's meta-algorithm (1994) combines principles of ESs and GAs in order to optimize a population of GAs, each of which is determined by 10 (continuous and discrete) components (in binary representation) representing both particular operator variants and parameter values. The meta-algorithm utilizes concepts from ESs to mutate a subset of the components, while the remaining components are mutated as in genetic algorithms according to a uniform probability distribution over the set of possible values. The objective function to be optimized by the base-level GAs was a simple sphere model. The average final best objective function value from two independent runs served as the fitness function for the meta-algorithm. The GAs found in the metaevolution experiment were considerably faster than standard GAs, and theoretical results about optimal mutation rates were confirmed in the experiments.

(vii) Lee and Takagi (1994) have presented a meta-GA-based approach for studying the effects of a dynamically adaptive population size, crossover, and mutation rate on De Jong's set of test functions (De Jong 1975). The base-level GAs had the same components as the ones used by Grefenstette (1986), but they were additionally augmented by a genetic representation of a fuzzy system, consisting of a fuzzy inference engine and a fuzzy knowledge base, to incorporate a variety of heuristic parameter control strategies into a single framework. The parameter settings for the meta-GA were fixed; a population size of 10 was used, and the meta-GA was run for 100 generations. The results indicated that a dynamic mutation rate contributed most to the on-line and off-line performance of the base-level GAs, again confirming recent theoretical results on optimal mutation rates.

(viii) Pham (1994) repeated Grefenstette's approach for different base-level objective functions as a preliminary step towards a proposal called *competitive evolution*. In this method, several populations, each with different operator variants and parameter settings, are allowed to evolve simultaneously. At each stage, the populations' performances are compared, and only the best populations, i.e. the one with the fittest member and the one with the highest improvement rate in the recent past, are allowed to evolve for a few more steps, then another comparison is made between all the populations. The competitive evolution method, however, may not be regarded as a metaevolutionary approach as described above; it is more a kind of *parallel (island)* population model with heterogeneous populations. C6.3

(ix) Tuson and Ross (1996a, b) have investigated the dynamic adaptation of GA operator settings by means of two different approaches. In their *coevolutionary* approach (Bäck 1996) (which is usually called *self-adaptive*) the operator settings are encoded into each individual of the GA population, and they are allowed to evolve as part of the solution process, without using a meta-GA. In the *learning rule adaptation* approach (Davis 1989, Julstrom 1995), information on operator performance is explicitly collected and used to adjust the operator probabilities. The effectiveness of both methods was investigated on a set of test problems. The results obtained with the coevolutionary approach on binary-encoded problems were disappointing, and the adaptation of operator settings by coevolution was found to be of little practical use. The results obtained using a learning-based approach were more promising, but still not satisfactory. The authors concluded that the adaptation mechanism should be separated from the main genetic algorithm and the information upon which decisions are made should explicitly be measured. This indicates that a metaevolutionary approach for parameter adaptation promises to be superior to the approaches investigated by Tuson and Ross—in contrast to many publications on ESs and EP where the benefits of self-adaptation could be demonstrated (Bäck 1996). C7.1

C7.2.7 Conclusions

In this section we have presented metaevolutionary approaches to determine the optimal evolutionary algorithm (EA), i.e. the best types of evolutionary operator and their parameter settings, for a given problem. The basic idea of a metaevolutionary approach is to consider the search for the best (base-level) EA as an optimization problem which is solved by a metalevel EA. We have presented an informal and formal description of the general metaevolutionary approach, pseudocode for realizing it, some theoretical results, and related work done in the area.

The metaevolutionary approaches proposed in the literature essentially differ in the way a base-level EA is encoded as a member of the metalevel population (depending on the base-level problem investigated and the number/type of operators and the parameter values to control them), the manner in which the types of operator and their parameter settings for the metalevel EA are determined, and the results obtained from the metaevolution experiments. A common feature is that metaevolutionary approaches usually require a high amount of computation time, but fortunately it is quite straightforward to develop a parallel implementation based on a manager–worker scheme (Bäck 1994, Freisleben and Härtfelder 1993a, b).

References

Bäck T 1993 Optimal mutation rates in genetic search *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 2–8

——1994 Parallel optimization of evolutionary algorithms *Parallel Problem solving from Nature—PPSN III (Proc. Int. Conf. on Evolutionary Computation and 3rd Conf. on Parallel Problem Solving from Nature, Jerusalem, October 1994)* (*Lecture Notes in Computer Science 866*) ed Y Davidor, H-P Schwefel and R Männer (Berlin: Springer) pp 418–27

——1996 *Evolutionary Algorithms in Theory and Practice* (New York: Oxford University Press)

Booker L 1987 Improving search in genetic algorithms *Genetic Algorithms and Simulated Annealing* ed L Davis (London: Pitman)

Davis L 1989 Adapting operator probabilities in genetic algorithms *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, June 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 61–9

——1991 *Handbook of Genetic Algorithms* (New York: Van Nostrand Reinhold)

De Jong K A 1975 *An Analysis of the Behaviour of a Class of Genetic Adaptive Systems* PhD Thesis, University of Michigan

Freisleben B and Härtfelder M 1993a In search of the best genetic algorithm for the traveling salesman problem *Proc. 9th Int. Conf. on Control Systems and Computer Science (Bucharest)* pp 485–93

——1993b Optimization of genetic algorithms by genetic algorithms *Proc. 1993 Int. Conf. on Artificial Neural Nets and Genetic Algorithms* ed R F Albrecht, C R Reeves and N C Steele (Vienna: Springer) pp 392–9

Freisleben B and Merz P 1996a A genetic local search algorithm for solving symmetric and asymmetric traveling salesman problems *Proc. 1996 IEEE Int. Conf. on Evolutionary Computation* (Piscataway, NJ: IEEE) pp 616–21

——1996b New genetic local search operators for the traveling salesman problem *Proc. 4th Int. Conf. on Parallel Problem Solving from Nature (Berlin, 1996)* (*Lecture Notes in Computer Science1141*) ed H-M Voigt, W Ebeling, I Rechenberg and H-P Schwefel (Berlin: Springer) pp 890–9

Glover F 1990 Tabu search: a tutorial *Interfaces* **20** 74–94

Goldberg D E 1989a *Genetic Algorithms in Search, Optimization and Machine Learning* (Reading, MA: Addison-Wesley)

——1989b Sizing populations for serial and parallel genetic algorithms *Proc. 3rd Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 70–9

Goldberg D E and Lingle R 1985 Alleles, loci and the travelling salesman problem *Proc. 1st Int. Conf. on Genetic Algorithms and their Applications (Pittsburgh, PA, July 1985)* ed J J Grefenstette (San Mateo, CA: Morgan Kaufmann) pp 154–9

Grefenstette J J 1986 Optimization of control parameters for genetic algorithms *IEEE Trans. Syst. Man Cybernet.* **SMC-16** 122–8

Hart W E and Belew R 1991 Optimizing an arbitrary function is hard for the genetic algorithm *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, July 1991)* ed R K Belew and L B Booker (San Mateo, CA: Morgan Kaufmann) pp 190–5

Hesser J and Männer R 1990 Towards an optimal mutation probability for genetic algorithms *Proc. 1st Int. Conf. on Parallel Problem Solving from Nature (Dortmund, 1990)* (*Lecture Notes in Computer Science 496*) ed H-P Schwefel and R Männer (Berlin: Springer) pp 23–32

Jog P, Suh J Y and van Gucht D 1989 The effects of population size, heuristic crossover and local improvement on a genetic algorithm for the traveling salesman problem *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, June 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 110–5

Julstrom B A 1995 What have you done for me lately? Adapting operator probabilities in a steady-state genetic algorithm *Proc. 6th Int. Conf. on Genetic Algorithms (Pittsburgh, PA, July 1995)* ed L J Eshelman (San Mateo, CA: Morgan Kaufmann) pp 81–7

Lee M A and Takagi H 1994 A framework for studying the effects of dynamic crossover, mutation, and population sizing in genetic algorithms *Advances in Fuzzy Logic, Neural Networks and Genetic Algorithms* (*Lecture Notes in Artificial Intelligence 1011*) ed T Furuhashi (Berlin: Springer) pp 111–26

Mercer R E and Sampson J R 1978 Adaptive search using a reproductive meta-plan *Kybernetes* **7** 215–28

Michalewicz Z 1994 *Genetic Algorithms + Data Structures = Evolution Programs* (Berlin: Springer)

Mühlenbein H and Schlierkamp-Voosen D 1995 Analysis of selection, mutation and recombination in genetic algorithms *Evolution and Biocomputation* (*Lecture Notes in Computer Science 899*) ed W Banzhaf and F H Eeckman (Berlin: Springer) pp 142–68

Nakano R, Davidor Y and Yamada T 1994 Optimal population size under constant computation cost *Parallel Problem solving from Nature—PPSN III (Proc. Int. Conf. on Evolutionary Computation and 3rd Conf. on Parallel Problem Solving from Nature, Jerusalem, October 1994)* (*Lecture Notes in Computer Science 866*) ed Y Davidor, H-P Schwefel and R Männer (Berlin: Springer) pp 130–8

Oliver I M, Smith D J and Holland J R C 1987 A study of permutation crossover operators on the travelling salesman problem *Proc. 2nd Int. Conf. on Genetic Algorithms (Pittsburgh, PA, July 1987)* ed J J Grefenstette (San Mateo, CA: Morgan Kaufmann) pp 224–30

Ostermeier A, Gawelczyk A and Hansen N 1994 Step-size adaptation based on non-local use of selection information *Parallel Problem solving from Nature—PPSN III (Proc. Int. Conf. on Evolutionary Computation and 3rd Conf. on Parallel Problem Solving from Nature, Jerusalem, October 1994)* (*Lecture Notes in Computer Science 866*) ed Yu Davidor, H-P Schwefel and R Männer (Berlin: Springer) pp 189–98

Pham Q T 1994 Competitive evolution: a natural approach to operator selection *Progress in Evolutionary Computation* (*Lecture Notes in Artificial Intelligence 956*) ed X Yao (Berlin: Springer) pp 49–60

Reeves C R 1993 *Modern Heuristic Techniques for Combinatorial Problems* (New York: Halsted)

Ros H 1989 Some results on Boolean concept learning by genetic algorithms *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, June 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 28–33

Schaffer J D, Caruna R A, Eshelman L J and Das R 1989 A study of control parameters affecting online performance of genetic algorithms for function optimization *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, June 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 51–60

Shahookar K and Mazumder P 1990 A genetic approach to standard cell placement using meta-genetic parameter optimization *IEEE Trans. Computer-Aided Design* **CAD-9** 500–11

Starkweather T, McDaniel S, Mathias K, Whitley C and Whitley D 1991 A comparison of genetic sequencing operators *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, July 1991)* ed R K Belew and L B Booker (San Mateo, CA: Morgan Kaufmann) pp 69–76

Tuson A L and Ross P 1996a Co-evolution of operator settings in genetic algorithms *Proc. 1996 AISB Workshop on Evolutionary Computing (Brighton, UK, April 1996)* ed T C Fogarty *(Lecture Notes in Computer Science 1143)* (New York: Springer) pp 120–8

——1996b Cost based operator rate adaptation: an investigation *Proc. 4th Int. Conf. on Parallel Problem Solving from Nature (Berlin, 1996)* (*Lecture Notes in Computer Science1141*) ed H-M Voigt, W Ebeling, I Rechenberg and H-P Schwefel (Berlin: Springer) pp 461–9

van Laarhoven P J M and Aarts E H L 1987 *Simulated Annealing: Theory and Applications* (Boston, MA: Kluwer)

White T and Oppacher F 1994 Adaptive crossover using automata *Parallel Problem solving from Nature—PPSN III (Proc. Int. Conf. on Evolutionary Computation and 3rd Conf. on Parallel Problem Solving from Nature, Jerusalem, October 1994)* (*Lecture Notes in Computer Science 866*) ed Yu Davidor, H-P Schwefel and R Männer (Berlin: Springer) pp 1229–38

D1.1 Overview of evolutionary computation as a mechanism for solving neural system design problems

V William Porto

Abstract

See the abstract for Chapter D1.

D1.1.1 Introduction

Although neural networks hold promise for solving a wide variety of problems, they have not yet fulfilled our expectations due to limitations in training, determination of the most appropriate topology, and efficient determination of the best feature set to use as inputs. A number of techniques have been investigated to solve these problems but none has the combination of simplicity, efficiency, and algorithmic elegance that is inherent in evolutionary computation (EC). These evolutionary techniques comprise a class of generalized stochastic algorithms which utilize the properties of a parallel and iterative search to solve a variety of optimization and other problems (see Sections B1.2, B1.3 and B1.4). EC is well suited to solving many of the inherently difficult or time-consuming problems associated with neural networks since most of the difficulties encountered with designing and training neural networks can be expressed as optimization problems.

B1.2, B1.3, B1.4

One of the most common problems encountered in training neural networks is the tendency of the training algorithm to become entrapped in local minima. This leads to suboptimal weight sets which often are insufficient to solve the task at hand. Due to the immense size of the typical search space, an exhaustive search is usually computationally impossible. Gradient methods, such as error backpropagation, are commonly used since these are easy to implement, may be tuned to provide superlinear convergence, and are mathematically tractable given the differentiability of the network nodal transfer functions. However these methods have the serious drawback that when the algorithm converges to a solution there is no guarantee that this solution is globally optimal. In real-world applications, these algorithms frequently converge to local suboptimal weight sets from which the algorithm cannot escape.

There is also the problem of determining the optimal network topology for the application. Much of the research attempting to provide optimal estimates of the number, interconnections, and types of nodes in the topology has been focused on bounding the solutions in a mean-squared-error (MSE) sense. The notion of nodal redundancy for robustness is often neglected, as is the fact that system performance may be better suited using a different metric for network topology determination (e.g. an asymmetric payoff matrix).

Finally, if one assumes that the aforementioned problems of network training and topology selection have been surmounted, there still remains the question of optimal input feature selection. Neural networks have been applied to a variety of problems ranging from pattern recognition to signal detection yet very little research has been made into ways to optimally select the most appropriate input features for each application. Typical approaches range from complex statistical measures to heuristic methodologies, each requiring *a priori* knowledge or specific tuning of the problem at hand.

Fortunately, stochastic EC methods not only can address the weight estimation and topology selection problem, but also can be utilized to help determine the optimal set of input features entering a neural

network. Searching and parameter optimization using stochastic EC methods can provide a comprehensive, self-adaptive solution to parameter estimation problems yet is often overlooked in favor of deterministic, closed-form solutions. The most general of these algorithms search the solution space in parallel, and as such are perfectly suited to application and implementation on today's multiprocessor computers.

D1.1.2 Conclusion

As will be seen in the next two sections, there exists a definite synergism between these two research areas. By combining the technologies of neural networks and EC, significant increases in functionality, trainability, and productivity can be made. The realization of a useful toolset is often dependent upon a cross-fertilization of research areas, as is evidenced by the examples shown. The reason that neural networks have been so popular is that they have been able to solve complex real-world problems which were previously addressable only by linearizing or seriously reducing the fidelity of the problem domain. Using EC methodologies in concert with neural networks will only serve to help further problem solving capabilities.

Further reading

There are several excellent general references available to the reader interested in furthering his or her knowledge in the area of EC and neural networks. The following books are a few well-written examples providing a good theoretical background into EC and neural network algorithms.

1. Bäck T 1996 *Evolutionary Algorithms in Theory and Practice* (New York: Oxford University Press)
2. Fogel D B 1995 *Evolutionary Computation, Toward a New Philosophy of Machine Intelligence* (Piscataway, NJ: IEEE)
3. Haykin S 1994 *Neural Networks, A Comprehensive Foundation* (New York: Macmillan)
4. Schwefel H-P 1981 *Numerical Optimization of Computer Models* (Chichester: Wiley)
5. Simpson P K 1990 *Artificial Neural Systems* (Elmsford, NY: Pergamon)
6. 1994 Special Issue on Evolutionary Computation *IEEE Trans. Neural Networks* **NN-5** No 1

D1.2 Evolutionary computation approaches to solving problems in neural computation

V William Porto

Abstract

See the abstract for Chapter D1.

D1.2.1 Training

The number of training algorithms and variations thereof recently published for different neural topologies is exceedingly large. The mathematical basis for the vast majority of these algorithms is to utilize gradient information to adjust the connection weights between nodes in the network. This is due to a certain mathematical tractability given the formulation of the training problem. Gradients of the error function are calculated and this information is propagated throughout the topology weights in order to estimate the best set of weights, usually in a least-squared-error sense (Werbos 1974, Rumelhart *et al* 1986, Hecht-Nielsen 1990, Simpson 1990, Haykin 1994, Werbos 1994). A number of assumptions about the local and global error surface are inherently made when using any of these gradient-based techniques. The existence of multiple extremum points is often neglected or overlooked entirely. Numerous modifications of simple gradient-based techniques have been made in order to speed up the often exceedingly slow convergence (training) rates. Stochastic training algorithms can provide an attractive alternative by removing many of these assumptions while simultaneously eliminating the calculation of gradients. Thus they are well suited for training in a wide variety of cases, and often perform better overall than the more traditional methods.

D1.2.1.1 Stochastic methods against traditional gradient methods

Traditionally, gradient-based techniques have provided the basic foundation for many of the neural network training algorithms (Rumelhart *et al* 1986, Simpson 1990, Sanchez-Sinencio and Lau 1992, Haykin 1994, Werbos 1994). It is important to note that gradient-based methods are used not only in training algorithms for feedforward networks (the most commonly known topology), but also in a variety of networks such as Hopfield nets, recurrent networks, radial basis function networks, and many self-organizing systems. Viewed within the mathematical framework of numerical analysis, gradient-based techniques often provide superlinear convergence rates in applications on convex surfaces. First-order (steepest- or gradient descent) and second-order (i.e. conjugate gradient, Newton, quasi-Newton) methods have been successfully used to provide solutions to the neural connection weight and bias estimation problem (Kollias and Anastassiou 1988, Kramer and Sangiovanni-Vincentelli 1989, Simpson 1990, Barnard 1992, Saarinen *et al* 1992). While these techniques may prove useful in a number of cases, they often fail due to several interrelated factors. First, by definition, in order to provide guaranteed convergence to a minimum point, first-order gradient techniques must utilize infinitesimally small step sizes (e.g. learning rates) (Luenberger 1973, Scales 1985). Step size determination is most often a balancing act between monotonic convergence and time constraints inherent in the available training apparatus. From a practical standpoint, training should be performed using the largest feasible step size to minimize computational time. Using a step size which is too large, however, may lead to a solution away from the desired optimum point even though the step is in the desired direction. Note that this is actually useful in escaping local minimum points

but must be implemented within a careful theoretical background (e.g. *simulated annealing*) to guarantee convergence. Several automated methods for step size determination have been researched with some providing near-optimal step size estimation (Jacobs 1988, Luo 1991, Porto 1993, Haykin 1994) in light of a certain class of problems. By the Kantorovich inequality, it can be shown that the method of the basic gradient descent algorithm converges linearly to a minimum point with a ratio no greater than $(\lambda_1 - \lambda_2)/(\lambda_1 + \lambda_2)$ where λ_1 and λ_2 are the largest and smallest eigenvalues, respectively, of the Hessian of the objective function evaluated at the solution point. However, convergence to a global optimum point is not guaranteed. D3.5

Second-order methods attempt to approximate the (inverse) Hessian matrix and utilize a line search for optimal step sizes at each iteration. These methods require the assumption that a reasonably smooth function in N dimensions can be approximated by a quadratic function over a small enough region in the vicinity of an optimal point. In both cases, however, the actual process of iteratively converging on the series of solutions is computationally expensive when viewed in a real-world sense where time is directly proportional to the number of floating-point (or integer) operations. For example, convergence of the Davidon–Fletcher–Powell (DFP) method is inferior to steepest descent with a step size error of only 0.1%, so, in terms of central processing unit (CPU) execution time, second-order information does not always provide superior convergence rates (Luenberger 1973, Shanno 1978). It should be noted that problems encountered when the Hessian matrix is indefinite or singular can be addressed by using the method of Gill and Murray, albeit with the added computational cost of solving a nontrivial-size set of linear equations (Luenberger 1973, Scales 1985). In practice, quasi-Newton methods work well only on relatively small problems with up to a few hundred weights (Dennis and Schnabel 1983).

One alternative approach to training neural networks is to utilize numerical solution of ordinary differential equations (ODEs) to estimate interconnection weights (Owens and Filkin 1989). By posing the weight estimation problem as a set of differential equations, ODE solvers can iteratively determine optimal weight sets. These methods, however, are subject to the same prediction–correction errors, and, in practice, these too can be quite costly computationally.

Hypothetically, one can find an optimal algorithm for determining step size with the desired gradient-based algorithm. A major problem still remains wherein all of the convergence theorems for these methods only prove convergence to an optimum point. There is no guarantee that this is the global optimum point except in the rare case where the function to be minimized is strictly convex. Research has proven convergence to a global optimum point is guaranteed on linearly separable problems when batch-mode processing (i.e. aggregation of errors prior to weight adjustment) error backpropagation learning is used (Gori and Tesi 1992). However, linearly separable problems are easily solved using non-neural-network methods such as linear discriminant functions (Fisher 1976, Duda and Hart 1973). In real-world applications, neural network training can and often does become entrapped in local minimum points, generating suboptimal weight estimates (Minsky and Papert 1988). The method most commonly used to overcome this difficulty is to restart the training process by using a different random starting point. Mathematically, restarting at different initial-weight-solution 'sample' points is actually an implementation of a simplistic stochastic process. However, this is a very inefficient search methodology.

Stochastic training methods provide an attractive alternative to the traditional methods of training neural networks. In fact, learning in Boltzmann machines is by definition probabilistic, and uses simulated annealing for weight adjustments. By their very nature, stochastic search methods, and evolutionary computation (EC) algorithms in particular, are not prone to entrapment in local minimum points. Nor are these algorithms subject to the step size problems inherent in virtually all of the gradient-based methods. As applied to the weight estimation problem, evolutionary methods can be viewed as sampling the solution (weight) space in parallel, retaining those weights which provide the best fitness score. Note that in an EC algorithm fitness does not necessarily imply a least-mean-squared-error criterion. Virtually any measure or combination of measures can be accommodated. In real-world environments robustness against failure of connections or nodes is often highly important. This robustness can easily be built into the networks during the training phase with EC training algorithms.

D1.2.1.2 Case studies

Evolutionary algorithms have been successfully applied to the aforementioned problem of training, i.e. estimating the optimal set of weights for neural networks. Numerous approaches have been studied, ranging from simple iterative evolution of weights to sophisticated schemes wherein recombination operators

exchange weight sets on subtrees in the topology. It is important to note that these algorithms typically do not utilize gradient information, and hence are often computationally faster due to their simplicity of implementation; fewer integer and/or floating-point operations are required.

Differences between several techniques suitable for training multilayer perceptrons (MLPs) and other neural networks were investigated by Porto (1992). The computational complexity of standard backpropagation (BP), modified (line search) BP, fast simulated annealing (FSA), and *evolutionary programming* (EP) were compared. In this paper, FSA using a Cauchy probability distribution for the annealing schedule is contrasted with EP. The EP weight optimization is performed with mutation variance inversely proportional to the root-mean-square (RMS) error of the aggregate input pattern training set. Thus the mutation variance decreases as training converges to more optimal solutions. Computational similarities between the FSA and EP approaches and increased robustness of a parallel search technique such as EP versus the single-solution member of an FSA search are shown. A number of tests are performed using underwater mine data using MLPs trained from multiple starting points with each of the aforementioned training techniques in order to ascertain the potential robustness of each to multimodal error surfaces. Results of this research on neural networks with multiple weight set solutions (i.e. local minima points) demonstrate better performance on naive test sets using FSA and EP training methods. These stochastic training methods are proven to be more robust to multimodal error surfaces and hence demonstrate reduced susceptibility to poor performance due to entrapment in local minima points. B1.4

The problem of robustness to processing node failure was addressed by Sebald and Fogel (1992). In their paper, adaptation of interconnection weights is performed with the emphasis on performance in the event of node failures. Neural networks are evolved using EP while linearly increasing the probabilistic failure rate of nodes. After training, performance is scored with respect to classification ability given N random failures during the testing of each network. Fault-tolerant networks are demonstrated as often performing poorly when compared against non-fault-tolerant networks if the probability of nodal failure is close to zero, but are shown to exhibit superior performance when failure modes are increased. EP is able to find networks with sufficient redundancy to be capable of dealing with nodal failure.

Using EC to evolve network interconnection weights in the presence of hardware weight value limitations and quantization noise was proposed by McDonnell (1992). A modified version of backpropagation is used wherein EP is used for estimating the solutions of bounded and constrained activation functions, and backpropagation is used to refine these solutions. Random competition of the weight sets is used to choose parent networks for each subsequent generation. Results of this research indicate the robustness of this technique and its wide range of applicability to a number of unconstrained, constrained, and potentially discontinuous nodal functions.

D1.2.2 Topology selection

Selection of an optimal topology for any given problem is perhaps even more important than optimizing the training technique. It is well known that suboptimal performance of any system can occur by overfitting of data using too many degrees of freedom (network nodes and interconnections) in the model. A balance must be struck between minimizing the number of nodes for generalization in learning and providing sufficient degrees of freedom to fully encode the problem to be learned while retaining robustness to failure. EC is well suited to this optimization problem, and provides for self-adaptive learning of overall topology as well.

D1.2.2.1 Traditional methodology against self-adaptive approaches

Selection of the most appropriate neural architecture and topology for a specific problem or class of problems is often accomplished by means of heuristic or bounding approaches (Guyon *et al* 1989, Haykin 1994). An eigensystem analysis via singular value decomposition (SVD) approach has been suggested by Wilson *et al* (1992) to estimate the optimal number of nodes and initial-starting-weight estimates in a feedforward topology. An SVD is performed on all patterns in the training set with the starting weights initialized using the unitary matrix. The number of nodes in the topology are determined as a function of the sigma matrix in a least-squares sense.

Other analytic and heuristic approaches have also been tried with some success (Sietsma and Dow 1988, Frean 1990, Hecht-Nielsen 1990, Bello 1992) but these are largely based upon probability distribution

assumptions, and presence of fully differentiable error functions. In practice, methods which are self-adaptive in determining the optimal topology of the network are the most useful as they are not constrained by *a priori* statistical assumptions. The search space of possible topologies is infinitely large, complex, multimodal, and not necessarily differentiable. EC represents a search methodology which is capable of efficiently searching this complex space.

D1.2.2.2 Case studies

Polani and Uthmann (1993) discuss the use of a *genetic algorithm* (GA) to improve the topology of Kohonen feature maps. In this study, a simple fitness function proportional to the measure of equidistribution of neuron weights is used. Flat network as well as toroidal and Mobius topologies are trained with a set of random input vectors. The GA tests show the existence of networks with non-flat topologies with the ability to be trained to higher-quality values than those expected for the optimal flat topology. Given that the optimally trainable topologies may lie distributed over different areas on the topological space, the GA approach is able to find these solutions without *a priori* knowledge and is self-adaptive. Use of this technique could prove valuable in construction of net topologies for selforganizing feature maps where convergence speed or adaptation to a given input space is crucial. B1.2

GAs are used to evolve the topology and weights simultaneously as described by Braun (1993). In weak encoding schemes, genes correspond to more abstract network properties, which are useful for efficiently capturing architectural regularities of large networks. However, strong encoding schemes require much less detailed knowledge about the genetic encoding and neural mechanisms. Braun researched a network generator capable of handling large real-world problems. A strong representation scheme is used where every gene of the genotype relates to one connection of the represented network. Once the maximal architecture is specified, potential connections within this architecture are chosen and iteratively mutated and selected. Crossover and mutation are performed using distance coefficients to prevent permuted interval representations in order to minimize connection length. This is where crossover alone often proves problematic. Tests on digit recognition, the truck-backer-upper task, and the Nine Men's Morris problem were performed. These experiments concluded that weight transmission from parent to offspring is very important and effectively reduces learning times. Braun also notes that mutation alone is potentially sufficient to obtain good performance.

As indicated previously, GAs generate new solutions by recombining representational components of two population members using a function known as crossover. Some degree of mutation is also used but the primary emphasis is on crossover. Specific task environments are characterized as *deceptive* when the fitness (goodness of fit) is not well correlated with the expected abilities inherent in its representational parts (Goldberg 1989, Whitley 1991). The deception problem is manifested in several ways. Note that identical networks (networks which share identical topologies and common weights when evaluated) need not have the same search representation since the interpretation function may be homomorphic. This leads to offspring solutions which contain repeated components. These offspring networks are often less fit than their parents, a phenomenon known as the competing conventions problem (Shaffer *et al* 1992). Second, the crossover operator is often completely incompatible with networks with different topologies. Finally, for any predefined task, a specific topology may have multiple solutions, each with a unique but different distribution of interconnections and weights. Since the computational role of each node is determined by these interconnections, the probability of generating viable offspring solutions is greatly reduced regardless of interpretation function. Fogel (1992) shows GA approaches are indeed prone to these deception phenomena when evolving connectionist networks. Efforts to reduce this susceptibility to deception are studied by Koza and Rice (1991); they utilize *genetic programming* (GP) techniques which generate neural networks with much more complex representations than traditional GA binary representations. They propose using these alternative representations in an effort to avoid interpretation functions which strongly bias the search for neural network solutions. B2.7.1 B1.5.1

The interpretation function which maps the search (representation) space to the evaluation (fitness) space in a GA approach will exceed the complexity of the learning problem (Angeline *et al* 1994). Recent trends have been focused away from binary representations in using GA approaches to solve neural network topology determination problems. Angeline proposes EP for connectionist neural network search as the representation evaluated by the fitness function is directly manipulated to produce increasingly more appropriate (better) solutions. The generalized acquisition of recurrent links (GNARL) algorithm (Angeline *et al* 1994) evolves neural networks using structural level mutations for topology selection

as well as simultaneously evolving the connection weights through mutation. Tests on a food tracking task evolved a number of interesting and highly fit solutions. The GNARL algorithm is demonstrated to simultaneously evolve the architecture and parameters with very little restriction of the architecture search space on a set of test problems.

The use of evolutionary search to determine the optimal distribution of radial basis functions (RBFs) was addressed by Whitehead and Choate (1994). Binary encoding is used in a GA with the evolved networks selected to minimize both the residual error in the function approximation as well as the number of RBF nodes. A set of space filling curves as encoded by the GA are evolved to optimally distribute the RBFs. The weights from the first layer, which form linear combinations of the RBFs, are trained with a conventional least-mean-squares learning rule. Convergence is rapid since the total squared error over the training set is a quadratic. An additional benefit is realized, wherein the local response of each RBF can be set to zero beyond a genetically selected radius, thus ensuring only a small fraction of the weights needs to be modified for each input training exemplar. This methodology strikes a balance between representations which specify all of the weights and requiring no training, and the other extreme where no weights are specified and full training of each network is required on each pass of the algorithm. Results indicate the superiority of evolving the RBF centers in comparison to k-means clustering techniques. This may possibly be explained by the fact that a large proportion of the evolved centers were observed to lie outside the convex hull of the training data, while the k-means clustering centers remained within this hull.

References

Angeline P, Saunders G and Pollack J 1994 Complete induction of recurrent neural networks *Proc. 3rd Annu. Conf. on Evolutionary Programming (San Diego, CA, 1994)* ed A V Sebald and L J Fogel (Singapore: World Scientific) pp 1–8

Barnard E 1992 Optimization for training neural networks *IEEE Trans. Neural Networks* **NN-3** 232–6

Bello M 1992 Enhanced training algorithms and integrated training/architecture selection for multilayer perceptron networks *IEEE Trans. Neural Networks* **NN-3** 864–75

Braun H 1993 Evolving neural networks for application oriented problems *Proc. 2nd Ann. Conf. on Evolutionary Programming (San Diego, CA, 1993)* ed D B Fogel and W Atmar (San Diego, CA: Evolutionary Programming Society) pp 62–71

Dennis J and Schnabel R 1983 *Numerical Methods for Unconstrained Optimization and Nonlinear Equations* (Englewood Cliffs, NJ: Prentice-Hall) pp 5–12

Duda R O and Hart P E 1973 *Pattern Classification and Scene Analysis* (New York: Wiley) pp 130–86

Fisher R A 1976 The use of multiple measurements in taxonomic problems *Machine Recognition of Patterns* (reprinted from 1936 *Annals of Eugenics*) ed A K Agrawala (New York: IEEE) pp 323–32

Fogel D B 1992 *Evolving Artificial Intelligence* PhD Dissertation, University of California

Frean M 1990 The upstart algorithm: a method for constructing and training feedforward neural networks *Neural Comput.* **2** 198–209

Goldberg D E 1989 *Genetic Algorithms in Search, Optimization and Machine Learning* (Reading, MA: Addison-Wesley) pp 1–54

Gori M and Tesi A 1992 On the problem of local minima in backpropagation *IEEE Trans. Pattern Anal. Machine Intell.* **PAMI-14** pp 76–86

Guyon I, Poujaud I, Personnaz L, Dreyfus G, Denker J and Le Cun Y 1989 Comparing different neural network architectures for classifying handwritten digits *Proc. IEEE Int. Joint Conf. on Neural Networks* vol II (Piscataway, NJ: IEEE) pp 127–32

Haykin S 1994 *Neural Networks, a Comprehensive Foundation* (New York: Macmillan) pp 121–281, 473–584

Hecht-Nielsen R 1990 *Neurocomputing* (Reading, MA: Addison-Wesley) pp 48–218

Jacobs R A 1988 Increased rates of convergence through learning rate adaptation *Neural Networks* **1** 295–307

Kollias S and Anastassiou D 1988, Adaptive training of multilayer neural networks using a least squares estimation technique *Proc. Int. Conf. on Neural Networks* vol I (New York: IEEE) pp 383–89

Koza J and Rice J 1991 Genetic generation of both the weights and architecture for a neural network *IEEE Joint Conf. on Neural Networks* vol II (Seattle, WA: IEEE) pp 397–404

Kramer A H and Sangiovanni-Vincentelli A 1989 Efficient parallel learning algorithms for neural networks *Advances in Neural Information Processing Systems 1* ed D S Touretzky (San Mateo, CA: Morgan Kaufmann) pp 40–8

Luenberger D G 1973 *Introduction to Linear and Nonlinear Programming* (Reading, MA: Addison-Wesley) pp 194–201

Luo Z 1991 On the convergence of the LMS algorithm with adaptive learning rate for linear feedforward networks *Neural Comput.* **3** 226–45

McDonnell J M 1992 Training neural networks with weight constraints *Proc. 1st Ann. Conf. on Evolutionary Programming (La Jolla, CA, 1992)* (San Diego, CA: Evolutionary Programming Society) pp 111–9

Minsky M L and Papert S A 1988 *Perceptrons* expanded edn (Cambridge, MA: MIT Press) pp 255–66

Owens A J and Filkin D L 1989 Efficient training of the back propagation network by solving a system of stiff ordinary differential equations *Proc. Int. Joint Conf. on Neural Networks* vol II (Piscataway, NJ: IEEE) pp 381–6

Polani D and Uthmann T 1993 Training Kohonen feature maps in different topologies: an analysis using genetic algorithms *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 326–33

Porto V W 1992 Alternative methods for training neural networks *Proc. 1st Ann. Conf. on Evolutionary Programming (La Jolla, CA, 1992)* ed D B Fogel and W Atmar (San Diego, CA: Evolutionary Programming Society) pp 100–10

——1993 *A Method for Optimal Step Size Determination for Training Neural Networks* ORINCON Internal Technical Report

Rumelhart D E and McClelland J (eds) 1986 *Parallel Distributed Processing: Explorations in the Microstructure of Cognition* vol 1 (Cambridge, MA: MIT Press) pp 318–30

Saarinen S, Bramley R B and Cybenko G 1992 Neural networks, backpropagation and automatic differentiation *Automatic Differentiation of Algorithms: Theory, Implementation and Application* ed A Griewank and G F Corliss (Philadelphia, PA: SIAM) pp 31–42

Sanchez-Sinencio E and Lau C (eds) 1992 *Artificial Neural Networks* (New York: IEEE) pp 2–32, 58–80, 94–106

Scales L E 1985 *Introduction to Non-Linear Optimization* (New York: Springer) pp 60–1

Sebald A V and Fogel D B 1992 Design of fault tolerant neural networks for pattern classification *Proc. 1st Ann. Conf. on Evolutionary Programming (La Jolla, CA, 1992)* ed D B Fogel and W Atmar (San Diego, CA: Evolutionary Programming Society) pp 90–9

Shaffer J D, Whitley D and Eshelman L J 1992 Combinations of genetic algorithms and neural networks: a survey of the state of the art *Proc. COGANN-92 Int. Workshop on Combinations of Genetic Algorithms and Neural Networks (Baltimore)* (Baltimore, MD: IEEE Computer Society) pp 1–37

Shanno D 1978 Conjugate-gradient methods with inexact searches *Math. Operat. Res.* **3** 244–56

Sietsma J and Dow R 1988 Neural net pruning—why and how *Proc. Int. Conf. on Neural Networks* vol I (New York: IEEE) pp 325–33

Simpson P K 1990 *Artificial Neural Systems* (Elmsford, NY: Pergamon) pp 90–120

Werbos P J 1974 *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences* PhD Thesis, Harvard University

——1994 *The Roots of Backpropagation from Ordered Derivatives to Neural Networks and Political Forecasting* (New York: Wiley) pp 29–81, 256–94

Whitehead B and Choate T 1994 Evolving space-filling curves to distribute radial basis functions over an input space *IEEE Trans. Neural Networks* **NN-5** 15–23

Whitley D 1991 Fundamental principles of deception in genetic search *Foundations of Genetic Algorithms* ed G Rawlins (San Mateo, CA: Morgan Kaufmann) pp 221–41

Wilson E, Umesh S and Tufts D 1992 Resolving the components of transient signals using neural networks and subspace inhibition filter algorithms *Proc. Int. Joint Conf. on Neural Networks (IJCNN '92, Baltimore)* vol I (Piscataway, NJ: IEEE) pp 283–8

D1.3 New areas for evolutionary computation research in neural systems

V William Porto

Abstract

See the abstract for Chapter D1.

D1.3.1 Introduction

There are many other areas in which the methodologies of evolutionary computation (EC) may be useful in the design and solution of neural network problems. Aside from training and topology selection, EC can be used to select optimal node transfer functions, which are often selected for their mathematical tractability, not their optimality in neural problems. Automated selection of input features is another area of current research with great potential. Evolving the optimal set of input features (from a potentially large set of transform functions) can be very useful in refining the preprocessing steps necessary to optimally solve a specific problem.

D1.3.2 Transfer function selection

One recent area of interest is the use of EC to optimize the choice of nodal transfer functions. Sigmoidal, Gaussian, and other functions are often chosen due to their differentiability, mathematical tractability, and ease of implementation. There exists a virtually unlimited set of alternative transfer functions ranging from polynomial forms and exponentials to discontinuous, nondifferentiable functions. By efficiently evolving the selection of these functions, potentially more robust neural solutions may be found. Simultaneous selection of nodal transfer functions and topology may be the ultimate evolutionary paradigm, as nature has taken this tack in evolving the brains of every living organism.

D1.3.3 Input feature selection

Evolutionary computation is well suited for automatically selecting optimal input features. By iterative selection of these input features for virtually any neural topology, evolutionary methods can be a more attractive approach than those of principal-components analysis and other statistical methods. Efficient, automatic search of this input feature space can significantly reduce the computational requirements of signal preprocessing and feature extraction algorithms.

Brotherton and Simpson (1995) devised an algorithm which automatically selects the optimal subset of input features and the neural architecture, as well as training the interconnection weights using *evolutionary programming* (EP). In developing a classifier for electrocardiogram (ECG) waveforms, EP was used to design a hierarchical network consisting of multilayer perceptrons (MLPs) for the first-layer networks, and fuzzy min–max networks for the second, output layer. The first-layer networks are trained and outputs fused in the second-layer network. EP is used to select from among several sets of input features. Initial training provided approximately 75% correct classification without including heart rate and phase features in the fusion network. Retraining of the fusion networks was performed with the EP trainer and feature selection mechanism, with the resulting system providing a 95% correct classification. Interestingly,

B1.4

analysis of the final trained network inputs showed the EP feature selection technique had determined that these two scalar input features had not been used, but had provided guidance during the training phase.

Chang and Lippmann (1991) examined the use of *genetic algorithms* (GAs) to determine the input B1.2 data and storage patterns and select appropriate features for classifier systems in both speech and machine vision problems. Using an EC approach they found they could reduce the input feature size from 153 features to only 33 features with no performance loss. Their investigations into solving a machine vision pattern recognition problem demonstrated the ability of GAs to evolve higher-order features which virtually eliminated pattern classification errors. Finally, in another of their tests with neural pattern classifiers, the number of patterns necessary to store was reduced by a factor of 8:1 without significant loss in performance.

The area of feature selection via EC will be of increased interest as more and more neural systems are put into the field. Selectively choosing the optimal set of input features can make the difference between a mere idea and a practical implementation.

References

Brotherton T and Simpson P 1995 Dynamic feature set training of neural networks for classification *Proc. 4th Ann. Conf. on Evolutionary Programming (San Diego, CA, 1995)* ed J R McDonnell, R G Reynolds and D B Fogel (Cambridge, MA: MIT Press) pp 79–90

Chang E and Lippmann R 1991 Using a genetic algorithm to improve pattern classification performance *Advances in Neural Information Processing Systems* ed D Touretsky (San Mateo, CA: Morgan Kaufmann) pp 797–803

D2.1 Technology and issues

C L Karr

Abstract

See the abstract for Chapter D2.

D2.1.1 Overview of technology

Rule-based systems, commonly called *expert systems*, have proven to be effective tools for solving a wide variety of practical problems (Waterman 1989). However, their performance in the area of *process control* F1.3 has lagged for several reasons including the uncertainty inherent in measuring and adjusting parameters in most process control environments. The uncertainty associated with process control, and with human decision-making, can be managed in expert systems that employ fuzzy set theory (Zadeh 1965). In fuzzy set theory, abstract concepts can be represented with *linguistic variables* (fuzzy terms), terms such as 'very high', 'fairly low,' and 'kind of fast'. The use of these fuzzy terms provides fuzzy logic controllers (FLCs) with a degree of flexibility generally unattainable in conventional rule-based systems used for process control.

FLCs have been used successfully in a number of simulated environments and in industrial control situations (Sugeno 1985, Evans *et al* 1989). These *fuzzy expert systems* include rules to direct the decision process and membership functions to convert linguistic variables into the precise numeric values needed by a computer for automated process control. The rule set is often gleaned from a human expert's knowledge (when such knowledge is available), which is generally based on his or her experience. Because linguistic terms are used in their construction, writing the rule set is often a straightforward task. Defining the fuzzy membership functions, on the other hand, is almost always the most time-consuming aspect of FLC design.

A standard method for determining the membership functions that produce maximum FLC performance would expedite FLC development. However, locating optimal membership functions is difficult because the performance of an FLC often changes dramatically due to small changes in either the membership functions or the rule set (Karr 1991a). Additionally, this task is even more difficult when it must be accomplished on-line as in an adaptive control system.

The adaptive capabilities, robust nature, and simple mechanics of evolutionary algorithms make them inviting tools for establishing the membership functions to be used in FLCs. Evolutionary algorithms will also be effective at generating the rules used in FLCs. Further, evolutionary algorithms possess qualities that will be beneficial in the design of adaptive FLCs which alter their rules and/or membership functions on-line to account for changes in the physical environment.

As far as the author is aware, neither *evolution strategies* nor *evolutionary programming* has been B1.3, B1.4 used to develop fuzzy–evolutionary systems (Kandel and Langholz 1994). *Genetic algorithms*, on the B1.2 other hand, have been used in the development of a number of effective systems by several researchers. The first article to address the synergy of genetic algorithms and fuzzy logic appeared in 1989 (Karr *et al* 1989). Subsequently, Karr and his coworkers adopted this approach to develop numerous fuzzy–evolutionary systems for engineering problems (Karr 1991a, b, Karr *et al* 1990, Karr and Gentry 1993a, b). The initial article acknowledged the difficulty of selecting membership functions that allowed for efficient FLC performance. A description was provided of an approach to membership function tuning that involved the use of a genetic algorithm. It did not take long for others to realize that there was a real need for methods of designing FLCs. Thrift (1991) also proposed the use of genetic algorithms for

designing FLCs. In his work, he suggested the use of genetic algorithms both for selecting the rule set and for tuning the membership functions. His approach was applied to a computer simulated translating cart (without the inverted pendulum), which is a bang–bang control problem. Results indicate that the genetic-algorithm-designed FLC approached the performance of an optimal controller.

Not surprisingly, several researchers have extended the approaches described by Karr and Thrift. Feldman (1993) proposed the use of a genetic algorithm for the synthesis of a fuzzy network. A fuzzy network is a connectionist extension of a fuzzy logic system allowing partially connected associations, or rules, that incorporate fuzzy terms. Fuzzy networks can be used to either model or control physical systems. In his paper, Feldman developed a fuzzy network to control a computer simulation of the very same translating cart system as presented by Thrift. Results indicate that the genetic-algorithm-designed fuzzy network is able to achieve a level of performance that is comparable with that of the FLC developed by Thrift. However, it is important to note that the fuzzy network required only six rules to achieve this performance level whereas Thrift's FLC utilized 18 rules. Thus, a genetic algorithm developed a fuzzy-logic-based control system that required a third of the rules needed in a human-developed fuzzy system for the same purpose. This is of course important because it substantially reduces the time necessary to arrive at an appropriate control action. Other researchers have since used genetic algorithms for tuning the membership functions in fuzzy, rule-based systems (Park and Kandel 1994, Wade *et al* 1994, Wu and Nair 1991).

Sun and Jang (1993) described the use of a genetic algorithm to design a fuzzy model of a medical diagnosis problem. The focus of this work was to use fuzzy logic not for a control strategy, but rather for a prediction strategy. In the development of their model, a genetic algorithm once again played a key role in designing the fuzzy logic system. Moreover, others have been actively researching the use of genetic algorithms for generating the rules to be used in fuzzy systems (Lee 1994, Lee and Smith 1994, Nishiyama *et al* 1992).

Homaifar and McCormick (1995) extended the work of Thrift by studying extensively the use of a genetic algorithm for simultaneously developing the rule set and tuning the membership functions associated with an FLC. In their paper, they argue that the performance of an FLC is dependent on the coupling of the rule set and the membership functions, and that therefore the two should not be developed independently of one another. Interestingly enough, their approach was applied to the development of an FLC for a cart–pole system. Their results indicate that a genetic algorithm is quite capable of generating a rule set while simultaneously tuning membership functions. However, whether or not the simultaneous designing of the rule set and the membership functions is vital is unclear.

All of the aforementioned efforts involve genetic algorithms in the role of function optimizers; fitness functions were written that gauged the performance of the potential fuzzy systems. Valenzuela-Rendon (1991) proposed an intriguing system in which a *classifier system* was provided with fuzzy logic capabilities. B1.5.2
A classifier system is a rule-based system that generates rules using a genetic algorithm. The classifier system rewards or punishes its rules based on their past performance, thereby creating the type of 'survival-of-the-fittest' environment a genetic algorithm needs to excel. His results indicate that there are situations in which the traditional classifier system needs the capabilities supplied by fuzzy logic. Additionally, the results indicate that there are a number of innovative ways to combine the search capabilities of genetic algorithms (or other evolutionary algorithms) with the approximate reasoning capabilities of fuzzy logic. In fact, classifier systems are similar in many ways to *genetic programming*. Thus, one would expect B1.5.1
to see applications of evolutionary programming to fuzzy–evolutionary systems in the future. The ideas introduced by Valenzuela-Rendon were implemented on a pH titration system by Karr and Phillips (1993).

Many control and modeling problems have values that change, and these changing parameters do not always appear directly in the rule set. Thus, when the values of these parameters change, the control system is unable to compensate. Karr and his coworkers have been successful in using a genetic algorithm for tuning and adapting FLCs on-line in response to changing values of parameters that do not appear explicitly in the fuzzy rule base (Karr and Gentry 1993a, Karr *et al* 1993). The development of adaptive systems is an area of current research in both fuzzy–evolutionary systems and neuroevolutionary systems (Goonatilake and Khebbal 1995).

The preceding citations are meant to provide the reader with a sampling of the work that has been done in the area of fuzzy–evolutionary systems. Although this list contains citations of the most important works in the area, it is by no means meant to be an all-inclusive review of the literature. The above review neglects a portion of work that is being continuously added to the literature. A number of excellent papers have not been mentioned. Two such papers come immediately to mind: (i) one by Surmann and coworkers

(1993) that addresses the synergism of genetic algorithms and fuzzy logic for a decision-support system, and (ii) a paper by Eick and Jong (1993) who combined fuzzy logic and genetic algorithms to produce a classification system. Additionally, a volume that addresses fuzzy–evolutionary systems exclusively is expected in the near future (Herrera and Verdegay 1996): it is likely that this volume will contain work in which evolutionary algorithms other than genetic algorithms are used to design fuzzy, rule-based systems.

D2.1.2 Issues in fuzzy-evolutionary system development

To date, only genetic algorithms have been used to design fuzzy, rule-based systems. However, no matter what the particular evolutionary algorithm selected, there are only three basic issues associated with the development of a fuzzy–evolutionary system. These issues follow closely the history of fuzzy–evolutionary system development outlined in the previous section, and are (i) selecting or tuning membership functions, (ii) choosing rules that will allow for the level of performance desired, and (iii) altering rules and/or membership functions in real time so that the number of rules required can be minimized without a sacrifice in performance.

Fuzzy, rule-based systems employ membership functions to allow a computer to manipulate linguistic terms such as 'high' and 'not very fast'. These membership functions work in conjunction with a rule set to provide the desired performance characteristics. Generally, a human expert is available to provide the rules needed to manipulate a particular environment, and these rules can be written relatively quickly. However, selecting the membership functions to be used in conjunction with the rule set is frequently a difficult task accomplished via trial and error. Evolutionary algorithms can be used to expedite the process of tuning the membership functions in a fuzzy, rule-based system.

Despite the fact that a human expert is available to provide rules for most problem environments, there are occasions when a rule set is simply not available. An example of such a system is a chaotic system in which a ball is bouncing on an oscillating table. The control objective is to adjust the frequency of oscillation of the table so that the ball always bounces to a constant height. In this problem environment, the height to which the ball bounces is extremely sensitive to the value selected for the frequency of oscillation. The chaotic nature of the system makes it extremely difficult for a human to write an effective rule set. However, determining an effective rule set is a search problem, and evolutionary algorithms effectively solve a wide variety of search problems. In fact a genetic-algorithm-designed fuzzy system has been developed that efficiently manipulates the chaotic ball–table system (Karr and Gentry 1993b).

One of the drawbacks associated with using fuzzy, rule-based systems for solving industrial-scale problems is that as the number of inputs to the fuzzy system increases the size of the rule base required increases multiplicatively. In the demanding problem environments often considered in industrial systems, numerous input variables are important. Thus, fuzzy systems result that contain very large numbers of rules. Karr and his coworkers (Karr and Gentry 1993a, Karr *et al* 1993) have suggested a technique in which the number of input variables considered can be reduced. The number of rules required for effectively manipulating a problem environment can be reduced if an evolutionary algorithm is used to alter the rules and membership functions employed by the fuzzy system in real time. The adaptive systems that result are less cumbersome because they employ a streamlined rule set, and can therefore respond more rapidly to the system being manipulated.

The above three issues of selecting membership functions, choosing rules, and altering both rules and membership functions in real time are the three major issues associated with fuzzy–evolutionary systems. To clarify these points, and to provide some detail as to how these three issues can be addressed with an evolutionary algorithm (specifically a genetic algorithm), the following Section D2.2 describes the steps D2.2 necessary to design an adaptive fuzzy control system for a particular problem. A genetic algorithm is used to develop an adaptive fuzzy–evolutionary system for the control of a cart–pole balancing system. The conclusions derived from that problem are presented below.

D2.1.3 Conclusions

FLCs have become increasingly viable solutions to process control problems. However, if the utility of these rule-based systems is to continue to grow, FLC development time must be decreased, and the difficulty associated with developing membership functions and writing the necessary rule sets must be reduced. Also, if FLCs are to provide practical solutions to complex process control problems, FLCs must be provided with adaptive capabilities. Fortunately, the robust search capabilities of evolutionary

algorithms make them viable tools for accomplishing the above-mentioned objectives, and, in fact, this chapter has demonstrated a genetic algorithm's ability to overcome many of the obstacles currently facing the development and implementation of fuzzy systems.

References

Eick C F and Jong D 1993 Learning Bayesian classification rules through genetic algorithms *Proc. 2nd Int. Conf. on Information and Knowledge Management* (Washington, DC, 1993)

Evans G W, Karwowski W and Wilhelm M R 1989 *Applications of Fuzzy Set Methodologies in Industrial Engineering* (Amsterdam: Elsevier)

Feldman D S 1993 Fuzzy network synthesis with genetic algorithms *Proc. 5th Int. Conf. on Genetic Algorithms* pp 312–7

Goonatilake S and Khebbal S (eds) 1995 *Intelligent Hybrid Systems* (Chichester: Wiley)

Herrera F and Verdegay J L (eds) 1996 *Genetic Algorithms and Soft Computing* (Heidelberg: Physica) in press

Homaifar A and McCormick E 1995 Simultaneous design of membership functions and rule sets for fuzzy controllers using GAs *IEEE Trans. Fuzzy Systems* in press

Kandel A and Langholz G (eds) 1994 *Fuzzy Control Systems* (Boca Raton, FL: Chemical Rubber Company)

Karr C L 1991a Genetic algorithms for fuzzy controllers *AI Expert* **6** 26–31

——1991b Applying genetics to fuzzy logic *AI Expert* **6** 38–43

Karr C L, Freeman L M and Meredith D L 1989 Improved fuzzy process control of spacecraft terminal rendezvous using a genetic algorithm *Proc. Intelligent Control and Adaptive Systems Conf.* vol 1196, pp 274–88

Karr C L and Gentry E J 1993a Fuzzy control of pH using genetic algorithms *IEEE Trans. Fuzzy Systems* **FS-1** 46–53

——1993b Control of a chaotic system using fuzzy logic *Fuzzy Control Systems* ed A Kandel and G Langholz (West Palm Beach, FL: Chemical Rubber Company) pp 475–97

Karr C L, Meredith D L and Stanley D A 1990 Fuzzy process control with a genetic algorithm *Control '90* (Society for Mining Metallurgy, and Exploration) pp 53–60

Karr C L and Phillips C J 1993 A fuzzy classifier system for process control *Proceedings of Technology 2003* vol 2, pp 7–16

Karr C L, Sharma S K, Hatcher W J and Harper T R 1993 Fuzzy control of an exothermic chemical reaction using genetic algorithms *Eng. Appl. Artificial Intell.* **6** 575–82

Lee M A 1994 *Automatic Design and Adaptation of Fuzzy Systems and Genetic Algorithms using Soft Computing Techniques* PhD Thesis, University of California, Davis

Lee M A and Smith M H 1994 Automatic design and tuning of a fuzzy system for controlling the acrobat using genetic algorithms, DSFS, and meta-rule techniques *Proc. 1994 Meeting North Am. Fuzzy Information Processing Soc.* pp 416–20

Nishiyama T, Takagi T, Yager R and Nakanishi S 1992 Automatic generation of fuzzy inference rules by genetic algorithm *Proc. 8th Fuzzy System Symp. (Hiroshima)* pp 237–40

Park D and Kandel A 1994 Genetic-based new fuzzy reasoning models with application to fuzzy control *IEEE Trans. Syst. Man Cybernet.* **SMC-24** 39–47

Sugeno M (ed) 1985 *Industrial Applications of Fuzzy Control* (Amsterdam: Elsevier)

Sun C and Jang J 1993 Using genetic algorithms in structuring a fuzzy rulebase *Proc. 5th Int. Conf. on Genetic Algorithms* p 655

Surmann H, Kanstein A and Goser K 1993 Self-organizing and genetic algorithms for an automatic design of fuzzy control and decision systems *Proc. EUFIT'93*

Thrift P 1991 Fuzzy logic synthesis with genetic algorithms *Proc. 4th Int. Conf. on Genetic Algorithms* pp 509–13

Valenzuela-Rendon M 1991 The fuzzy classifier system: a classifier system for continuously varying variables *Proc. 4th Int. Conf. on Genetic Algorithms* pp 346–53

Wade R L, Walker G W and Phillips C 1994 Combining genetic algorithms and aircraft simulations to tune fuzzy rules in a helicopter control system *Conf. on Advances in Modeling and Simulation (Huntsville, AL, 1994)*

Waterman D A 1989 *A Guide to Expert Systems* (Reading, MA: Addison-Wesley)

Wu K and Nair S S 1991 Self organizing strategies for fuzzy control *Proc. North Am. Fuzzy Information Processing Soc. 1991 Workshop* pp 296–300

Zadeh L A 1965 Fuzzy sets *Information Control* **8** 338

D2.2 A cart–pole system

C L Karr

Abstract

See the abstract for Chapter D2.

D2.2.1 Introduction

This article describes the control of a cart–pole balancing system. A cart is free to translate along a one-dimensional track while a pole is free to rotate only in the vertical plane of the cart and track. A multivalued force, F, is applied at discrete time intervals to the center of mass of the cart. A schematic of the cart–pole system is shown in figure D2.2.1(a). The objective of the control problem is to apply forces to the cart until it is motionless at the center of the track and the pole is balanced in a vertical position. A block diagram of the control loop is shown in figure D2.2.1(b). This task of centering a cart on a track while balancing a pole is often used as an example of the inherently unstable, multiple-output, dynamic systems present in many balancing situations, such as two-legged walking and the stabilization of a rocket thruster.

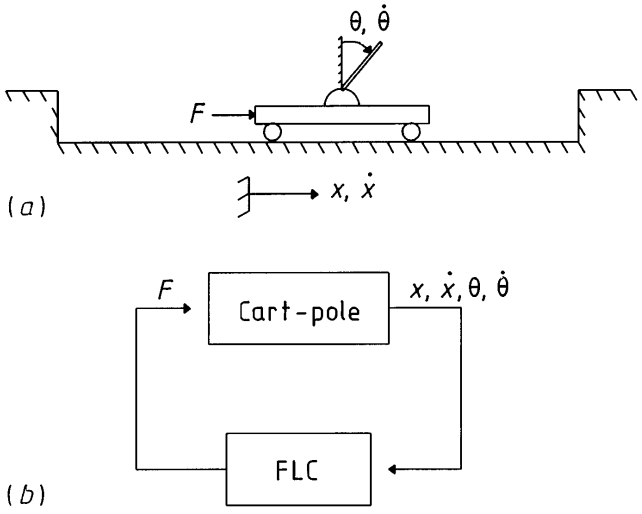


Figure D2.2.1. (a) Cart–pole system, (b) control loop.

The state of the cart–pole system at any time is described by four real-valued state variables:

x = position of the cart
$\dot{x}$ = linear velocity of the cart
θ = angle of the pole with respect to the vertical
$\dot{\theta}$ = angular velocity of the pole.

The system is modeled by the following nonlinear ordinary differential equations (Barto *et al* 1983):

$$\ddot{\theta} = \frac{g \sin\theta + \cos\theta \left[\dfrac{-F - m_p l \dot{\theta}^2 \sin\theta + \mu_c \operatorname{sign}(\dot{x})}{(m_c + m_p)} \right] - \dfrac{\mu_p \dot{\theta}}{m_p l}}{l \left[\dfrac{4}{3} + \dfrac{m_p \cos^2\theta}{(m_c + m_p)} \right]} \tag{D2.2.1}$$

$$\ddot{x} = \frac{F + m_p l [\dot{\theta}^2 \sin\theta - \ddot{\theta} \cos\theta] - \mu_c \operatorname{sign}(\dot{x})}{(m_c + m_p)} \tag{D2.2.2}$$

where

$g = 9.81$ m s^{-2}, acceleration due to gravity
$m_c = 1.0$ kg, mass of cart
$m_p = 0.1$ kg, mass of pole
$l = 0.5$ m, length of pole
$m_c = 0.0005$, coefficient of friction of cart on track
$m_p = 0.000\,002$, coefficient of friction of pole on cart
-10.0 N $< F < 10.0$ N, force applied to cart's center of mass.

The solution of these equations was approximated using Euler's method, thereby yielding the following difference equations:

$$\dot{\theta}^{t+1} = \dot{\theta}^t + \ddot{\theta}^t \, \Delta t \tag{D2.2.3}$$

$$\theta^{t+1} = \theta^t + \dot{\theta}^t \, \Delta t \tag{D2.2.4}$$

$$\dot{x}^{t+1} = \dot{x}^t + \ddot{x}^t \, \Delta t \tag{D2.2.5}$$

$$x^{t+1} = x^t + \dot{x}^t \, \Delta t \tag{D2.2.6}$$

where the superscripts indicate values at a particular time, Δt is the time step, and the values $\ddot{x}^t$ and $\ddot{\theta}^t$ are evaluated using equations (D2.2.1) and (D2.2.2). A time step of 0.02 seconds was used because this time step struck a balance between the accuracy of the solution and the computational time required to find the solution and because it was suggested by Barto *et al* (1983).

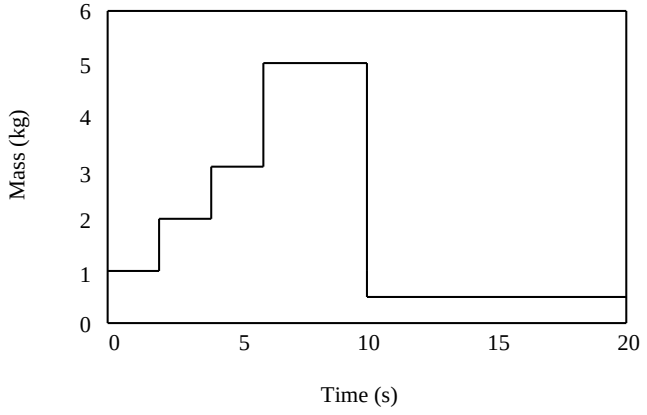


Figure D2.2.2. Changing mass complicates the problem.

The preceding is a description of the classic cart–pole system as it is generally addressed in the literature. The characteristic parameters of the physical system, parameters such as cart mass and pole length, remain constant. However, to demonstrate the issue of real-time adaptation of a fuzzy logic controller (FLC), the control problem is made considerably more difficult by allowing the mass of the cart to change with time as shown in figure D2.2.2. This expansion transforms the problem into a time-varying control problem. Note that the cart mass increases by a factor of up to five, which significantly alters the

response of the cart–pole system to a given force stimulus. To keep the size of the rule set required by the FLC to a minimum and to reduce the computation time needed by the FLC to select an appropriate action, the mass of the cart is not included in the rule set. Therefore, changes in the response of the cart–pole system must be accounted for by altering the membership functions in real time (or by altering the rule set, an alternative that is not covered in this presentation). Thus, an adaptive FLC is required, one that is able to account for changes in variables that do not explicitly appear in the controller's rule set.

D2.2.2 Evolutionary design of a fuzzy controller

There are numerous approaches to developing FLCs. Unfortunately, a large number of these approaches are complex and utilize cumbersome *fuzzy mathematics* that is simply not needed to implement a fuzzy control system. In this section, a basic approach to the development of an FLC is presented. A step-by-step procedure for fuzzy control of the cart–pole system is provided. This procedure is written in a general form so that it may be easily adapted to the development of other FLCs.

The first step in developing the cart–pole FLC is to determine which variables will be important in choosing an effective control action. These variables are used to calculate errors and changes in error that appear on the left side of the rules which are of the form: IF {condition} THEN {action}. In the cart–pole balancing system, the four state variables listed in the previous Section D2.1 control the system. Thus, D2.1 there will be two error terms: $E_x = s_x - x$ and $E_\theta = s_\theta - \theta$ where s_x and s_θ are the respective setpoints for cart position and pole angle ($s_x = s_\theta = 0.0$). There will also be two change-in-error terms: $\Delta E_x = \dot{x}$ and $\Delta E_\theta = \dot{\theta}$. In a 'common sense' approach, any decision on the action to be taken (the magnitude and direction of the force to be applied to the cart) must be based on the current value of these four variables. Next, a determination must be made as to what specific actions can be taken on the system. In the cart–pole balancing system, the only action variable is the value of the force, F, to be applied to the center of mass of the cart.

The second step in the design of an FLC is the selection of linguistic terms to represent each of the variables. There is no unique method of doing this: the number and definition of the linguistic terms is always problem specific, and requires a general understanding of the system to be controlled. For this application, four linguistic terms were used to describe the error associated with the position of the cart, E_x. Three linguistic terms were deemed adequate to represent the variables, ΔE_x, E_θ, and ΔE_θ. The variable, F, necessitated the use of seven linguistic terms for adequate representation. The specific linguistic terms used to describe the variables follow:

E_x	Negative Big (**NB**), Negative Small (**NS**), Positive Small (**PS**) and Positive Big (**PB**)	
ΔE_x	Negative (**N**), Near Zero (**NZ**), and Positive (**P**)	
E_θ	Negative (**N**), Near Zero (**NZ**), and Positive (**P**)	(D2.2.7)
ΔE_θ	Negative (**N**), Near Zero (**NZ**), and Positive (**P**)	
F	Negative Big (**NB**), Negative Medium (**NM**), Negative Small (**NS**), Zero (**Z**), Positive Small (**PS**), Positive Medium (**PM**), and Positive Big (**PB**).	

The third step in the design of an FLC is to 'define' the linguistic terms using *membership functions*. As with the initial requirement of selecting the necessary linguistic terms, there are no definite guidelines for constructing the membership functions: the terms are defined to represent the designer's general conception of what the terms mean. Membership functions can come in virtually any form. Two commonly used membership function forms (and the two forms used here) are triangular and trapezoidal. The only restriction generally applied to the membership functions is that they have a maximum value of 1 and a minimum value of 0. When a membership function value is 1, there is complete confidence in the premise that the value of a variable is completely described by the particular linguistic term. When a membership function value is 0, there is complete confidence in the premise that the value of a variable is *not* described by the particular linguistic term. It is important to select membership functions that portray the developer's and the potential user's general conception of the linguistic terms. A genetic algorithm will be used to refine the membership functions to provide for near-optimal FLC performance. The membership functions developed by the authors for the cart–pole balancer appear in figure D2.2.3. Note that in the membership functions for E_x the left-most triangle is the fuzzy set for NB, the two isosceles triangles represent NS and PS, and the right-most triangle represents PB. For ΔE_x the left-most triangle represents N, the middle

triangle represents NZ, and the right-most triangle represents P. For E_θ the left-most triangle represents N, the middle triangle represents NZ, and the right-most triangle represents P. For ΔE_θ the left-most triangle represents N, the middle triangle represents NZ, and the right-most triangle represents P.

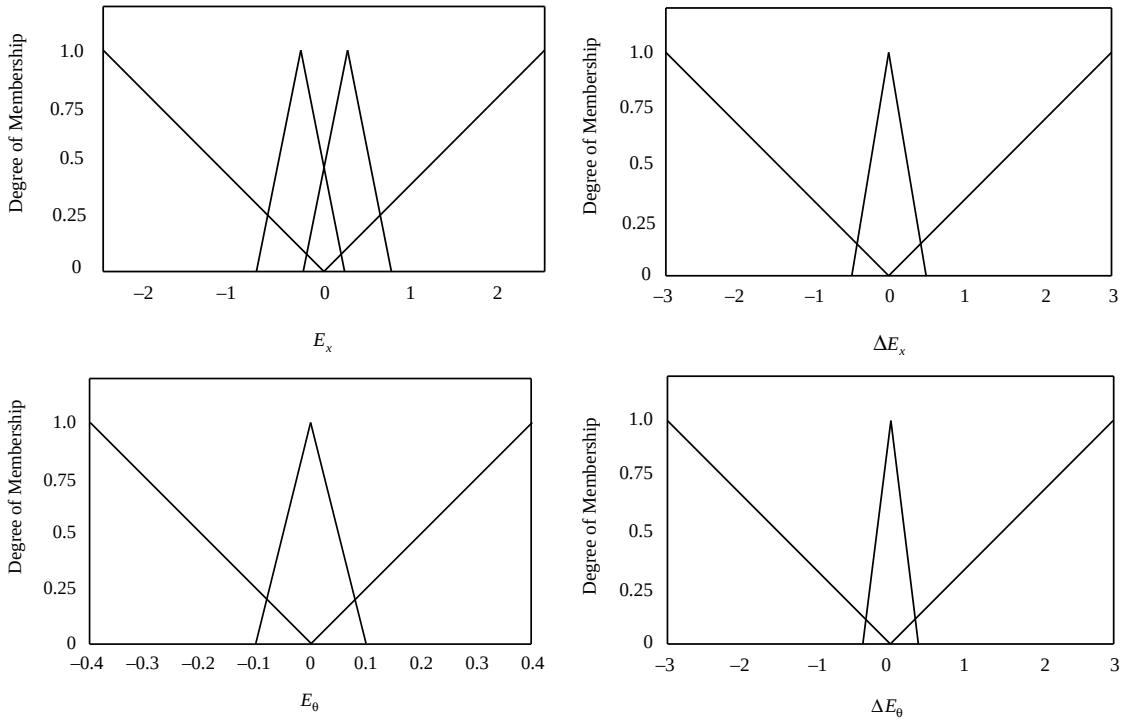


Figure D2.2.3. Author-developed membership functions.

The fourth step in the design of an FLC is the development of a rule set. The rule set in an FLC must include a rule for every possible combination of the variables as they are described by the chosen linguistic terms. Thus, 108 rules are required for the cart–pole balancing FLC as designed to this point ($4 \times 3 \times 3 \times 3 = 108$ possible combinations of the variables E_x, ΔE_x, E_θ, and ΔE_θ). Due to the nature of the linguistic terms, many of the actions needed for the 108 possible condition combinations are readily apparent. For instance, when the position of the cart is NB, the velocity of the cart is N, the position of the pole is P, and the angular velocity of the pole is P, the required action is without a doubt to apply a PB force to the cart. However, there are some conditions for which the appropriate action is not readily apparent.

In fact, there are some conditions for which the selection of an appropriate action seems almost contradictory. As an example, what is the appropriate action when the position of the cart is PS, the velocity of the cart is P, the position of the pole is NZ, and the angular velocity of the pole is P? The cart is to the right of the centerline and moving further away from the setpoint. Thus, if one is considering only the cart, the appropriate action would be to apply a small force in the negative direction. However, obviously one cannot consider only the cart. The state of the pole requires a small force in the positive direction. The traditional way to resolve these conflicts and to select an appropriate action has been to experiment with different selections of the action variables. An alternative approach is to allow a genetic algorithm to select an effective rule set for the membership functions as they have been defined by the developer. The rule set that is used for the cart–pole balancer was acquired the 'old-fashioned way': trial and error directed by experience in controlling the system. The complete rule set used for the cart–pole balancing system appears in figure D2.2.4.

Now that both the controller variables have been chosen and described with linguistic terms, and a rule set has been written that prescribes an appropriate action for every possible set of conditions, it remains to determine a single value for the force to be applied to the cart at a particular time step. This is a concern because more than one of the 108 possible rules can be applicable for a given state of the cart–pole system. A common technique for accomplishing this task is the center of area method (Sugeno

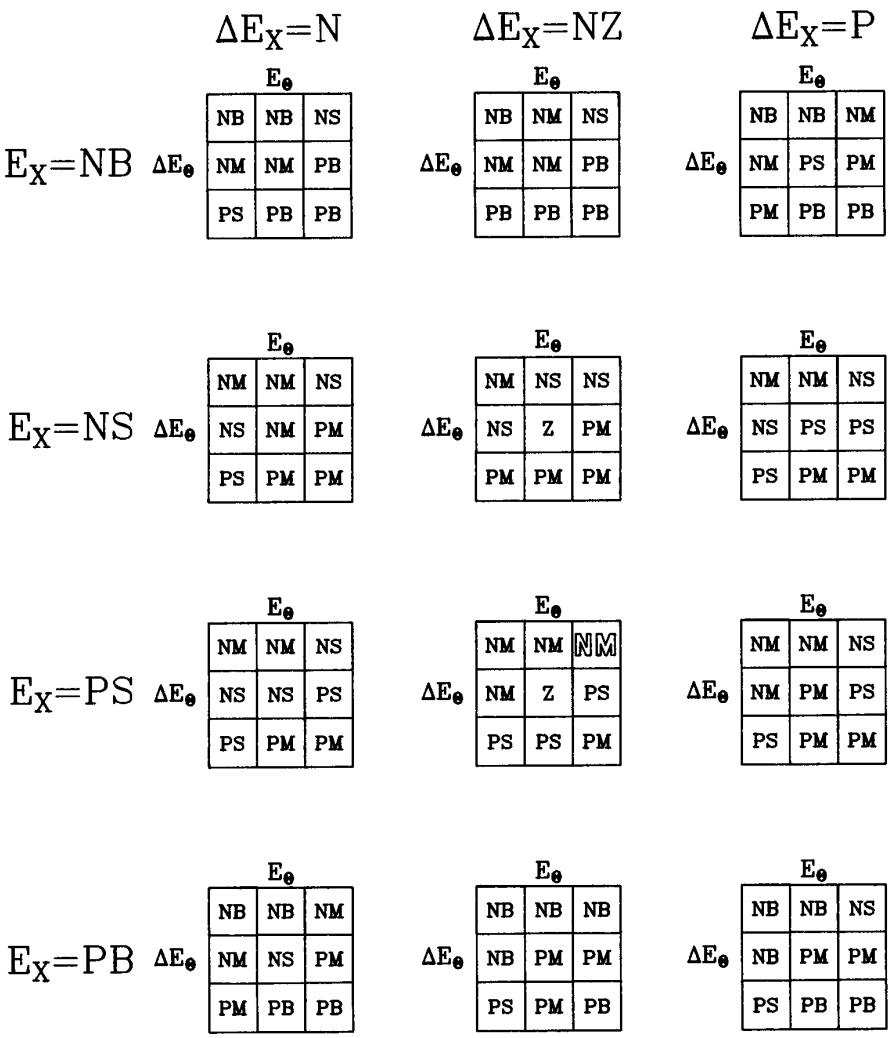


Figure D2.2.4. Rule set for the cart–pole system.

1985) (sometimes called the *centroid method*). In the center of area method, the action prescribed by each rule plays a part in the final value of F. The contribution of each rule to the final value of F is proportional to the minimum confidence (the minimum value of the membership function values on the left side of the rule) one has in that rule for the specific state of the physical system at the particular time. This is equivalent to taking a weighted average of the prescribed actions. With the determination of a strategy for resolving 'conflicts' in the actions prescribed by the individual rules, the FLC is complete.

The step-by-step procedure described above for developing an FLC is summarized below:

(i) determine the variables to be considered;
(ii) select linguistic terms to represent each variable;
(iii) 'define' the linguistic terms using membership functions;
(iv) establish a set of fuzzy production rules that cover all of the possible conditions that could exist in the problem environment.

D2.2.2.1 The role of the evolutionary algorithm

As described in the preceding chapters of this book, there are three particular evolutionary algorithms. Of these, only *genetic algorithms* have been used in the development of fuzzy–evolutionary systems. There are numerous flavors of genetic algorithms; several genetic operators and variations of the basic scheme B1.2

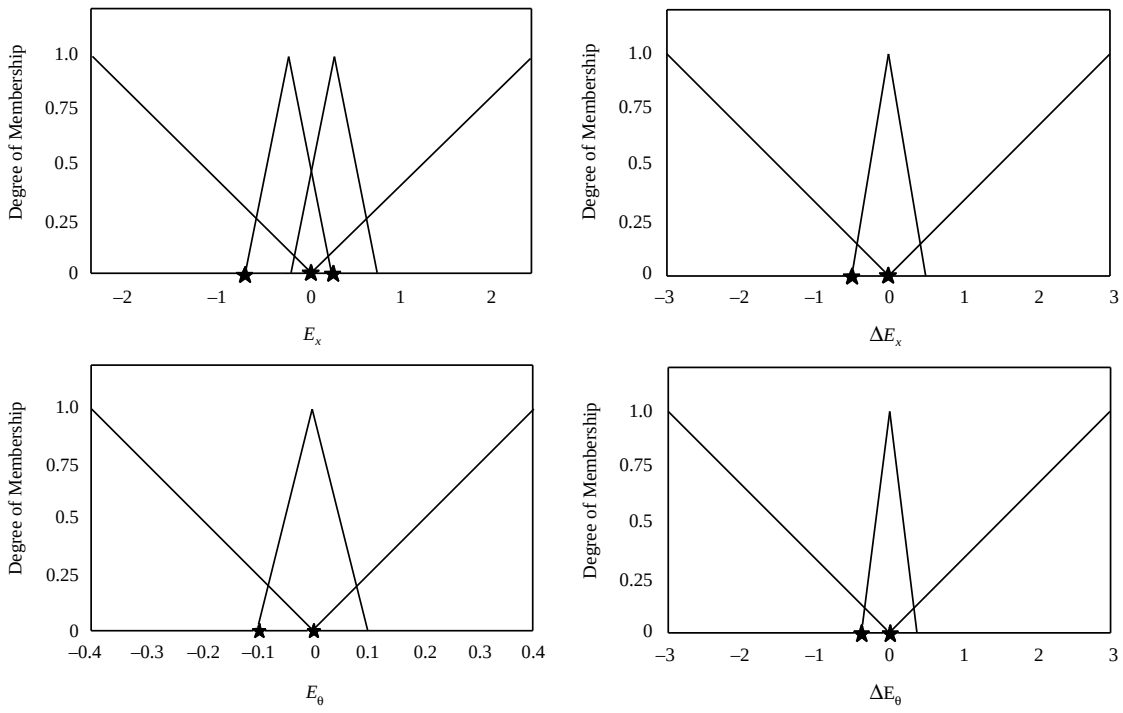


Figure D2.2.5. Nine parameters to be determined.

have been developed and implemented. Once the details of the particular genetic algorithm to be employed have been determined, there are basically two decisions to be made when utilizing a genetic algorithm to select FLC membership functions or rule sets: (i) how to code the possible choices of membership functions or rules as finite bitstrings and (ii) how to evaluate the performance of the FLC composed of the chosen membership functions and rules.

Since the issues associated with membership function definition and rule selection are quite similar, only the search for membership functions is investigated. Consider the selection of a coding scheme. To define an entire set of triangular membership functions (functions for E_x, ΔE_x, E_θ, ΔE_θ, and F), several parameters must be selected. First, make the distinction between the two types of triangle used (see figure D2.2.3). The right (90°) triangles appearing on the left and right boundaries will be termed *extreme* triangles, while the isosceles triangles appearing between the boundaries will be termed *interior* triangles. Only one point must be specified to completely define an extreme triangle, because the apex of the triangle is fixed at the associated extreme value of the condition or action variable (the maximum value of NB for the error in cart position will always be at $E_x = -2.4$). On the other hand, the complete definition of an interior triangle necessitates the specification of two points, given the constraint that the triangles must be isosceles, i.e. the apex is at the midpoint of the two points specified. Thus for the complete definition of a set of triangular membership functions for the cart–pole balancer as described above, 30 points must be specified (6 for E_x; 4 for E_x, ΔE_x, and E_θ; and 12 for ΔE_θ).

The search space associated with the selection of membership functions for the cart–pole balancer can be pruned from its original 30-parameter form. Notice the rule set is symmetric because every condition wherein the cart is to the left of the track's center has an analogous condition wherein the cart is to the right of the track's center. Therefore, NB should be the 'opposite' of PB, NS should be the 'opposite' of PS, and so on for all of the membership functions. Thus, instead of finding 30 parameters, the genetic algorithm is faced with the task of finding only the nine points identified in figure D2.2.5. Although the original search space has been reduced substantially, a nine-parameter search problem can still be of some consequence.

Now that the pertinent search parameters have been identified, a strategy for representing a set of these parameters as a finite bitstring must be developed. One such strategy that is popular, flexible, and effective is *concatenated, mapped, unsigned binary coding*. In this coding scheme each individual parameter is discretized by mapping linearly from a minimum value ($C_{\min}$) to a maximum value ($C_{\max}$)

using a 4-bit (although the length of the substrings does not have to be fixed at 4), unsigned binary integer according to the equation

$$C = C_{\min} + \frac{b}{(2^l - 1)}(C_{\max} - C_{\min}) \tag{D2.2.8}$$

where C is the value of the parameter of interest, and b is the decimal value represented by an l-bit string. Representing more than one parameter (such as the nine parameters necessary in the cart–pole balancer) is accomplished simply by concatenating the individual 4-bit segments. Thus, in this example, a 36-bit string is necessary to represent an entire set of membership functions. This discretization of the problem produces a search space in which there exist $2^{36} = 6.872 \times 10^{10}$ possible solutions. It is important to note that unlike the case with a genetic algorithm, the problem of coding is not an issue in the implementation of evolutionary computing. The code, however, reduces the problem to a grid search problem. Using evolution strategies or evolutionary programming, it remains a continuous problem with much better resolution of the search space.

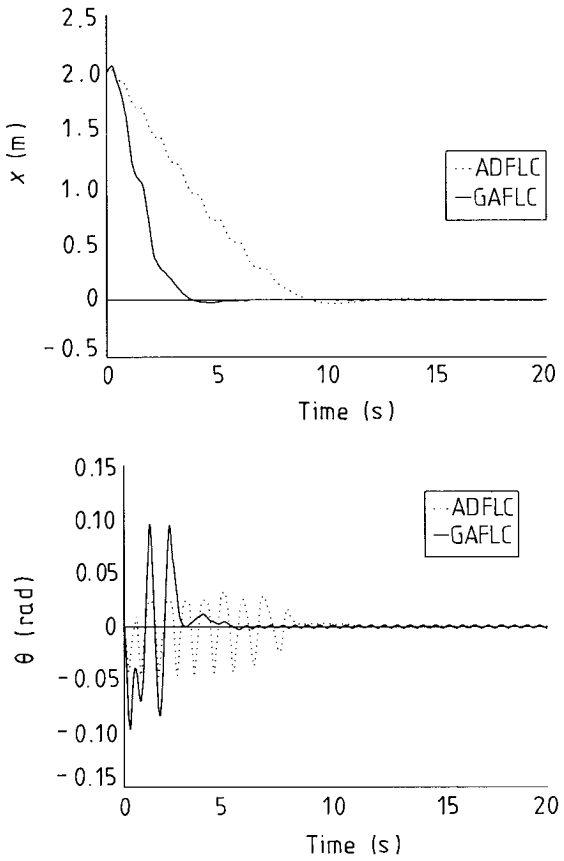


Figure D2.2.6. Cart–pole balancer: constant mass.

The second decision that must be made is to determine how the strings, or the potential membership functions, are to be evaluated. In judging the performance of the cart–pole FLC, it is important for the controller to center the cart and to balance the pole. It should accomplish these tasks in the shortest time possible when initiated from any of a number of different initial conditions. These two objectives, centering the cart and balancing the pole, can be achieved by designing the rules and membership functions used in the FLC to minimize both a weighted sum of the absolute value of the distance between the cart and the center of the track and the absolute value of the difference between the angle of the pole and vertical. The actual objective function the genetic algorithm minimized in this study is

$$f = \sum_{i=\text{case 1}}^{i=\text{case 4}} \sum_{j=0\,\text{s}}^{j=\max\ \text{time}} (w_1|(x_{ij}| + w_2|\theta_{ij}|) \tag{D2.2.9}$$

where $w_1 = 1.0$ and $w_2 = 10.0$ are weighting constants selected to weight the objectives associated with cart position and pole angle equally (values of x are on the order of ten times the magnitude of the values of θ), the four cases are four different sets of initial conditions for the cart–pole system, and max time is 30 seconds which is a reasonable period for accomplishing the control objectives. Four initial-condition cases were considered to ensure the FLC could accomplish the objective of centering the cart while balancing the pole from different starting points.

D2.2.3 Results

Two sets of results are presented to demonstrate the efficacy of using a genetic algorithm to select membership functions. First, results for the traditional cart–pole balancing system are given. Next, results are presented in which a genetic algorithm is used to select membership functions on-line for the time-varying cart–pole balancing system.

Figure D2.2.6 shows results for the cart–pole balancer in which the mass of the cart remains constant. The author-developed FLC (ADFLC) is compared to a FLC that uses membership functions selected by a genetic algorithm (GAFLC). The GAFLC is able to achieve the goal of centering the cart and balancing the pole in approximately 7 seconds as compared to the 20 seconds required by the ADFLC. Therefore, a genetic algorithm has improved the initial design of the FLC. The question of whether or not this technique can be used to alter membership functions in real time remains unanswered.

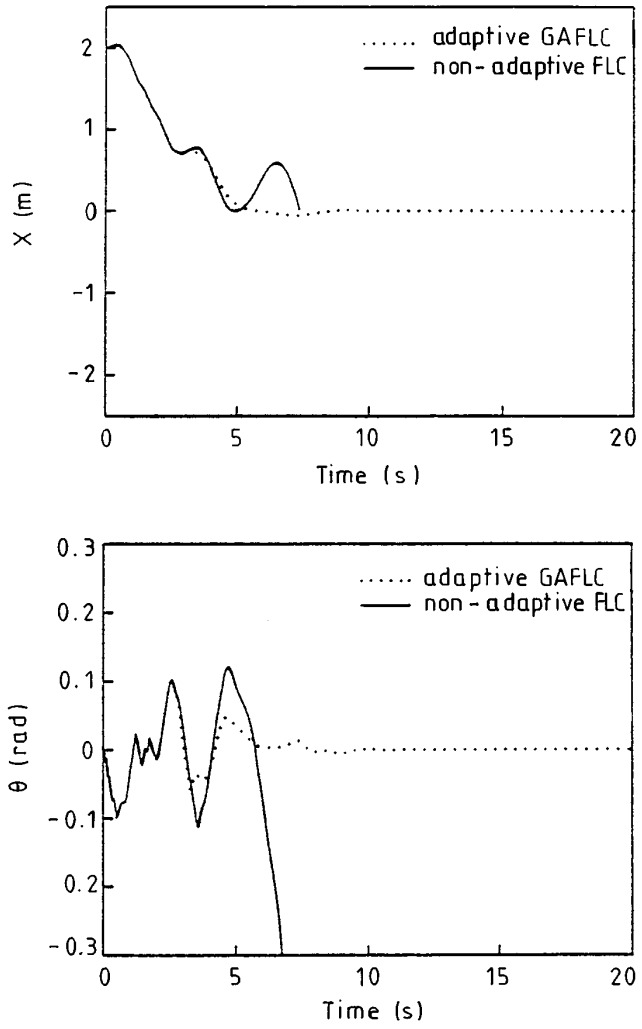


Figure D2.2.7. Cart–pole balancer: mass changing.

When the mass of the cart changes with time, the cart–pole balancing system responds differently to the application of forces. However, the mass of the cart was intentionally left out of the rule set because the inclusion of additional parameters increases the size of the rule set multiplicatively. The basic approach to using a genetic algorithm to select high-performance functions as outlined in the preceding sections is not modified by the introduction of a time-varying parameter. However, some of the details needed to implement the technique are slightly different. First, there is no need to alter the membership functions unless the mass of the cart changes. In the results presented, every time a change in mass occurs, a genetic algorithm begins a search for new membership functions. Second, there is no need to look for robust membership functions that can accomplish the control objective from any set of initial conditions because the FLC must be able to achieve the control objective beginning from the current state of the system. Therefore, the objective function does not have to include a summation over four initial-condition cases, and the function evaluations associated with a genetic algorithm are faster by a factor of four.

Figure D2.2.7 shows the results of an adaptive GAFLC that accounted for changes in the cart mass. This adaptive GAFLC was able to avoid the catastrophic failures of the pole falling over or the cart striking a wall despite the dramatic changes in cart mass (seen in figure D2.2.2) by doing nothing more than altering its membership functions on-line. Every time a change in cart mass was made, a genetic algorithm was employed to locate new membership functions that were effective for the current state of the system. As can be seen in figure D2.2.7, the adaptive GAFLC outperformed a non-adaptive FLC.

The preceding has been an exposition on the use of a genetic algorithm to improve the performance of an FLC both in the initial design phase and in the real-time adaptation of a controller. The emphasis has been on the steps necessary to address the three major issues associated with the design of fuzzy–evolutionary systems. Although the focus was on the alteration of membership function values, the rules could have as easily been the focus of the search problem. In such a case, 3-bit substrings could have been used to represent the seven possible values for each of the 108 rules needed. Thus, a 324-bit string would be used, and the fitness function would remain unchanged.

References

Barto A G, Sutton R S and Anderson C W 1983 Neuronlike adaptive elements that can solve difficult learning control problems *IEEE Trans. Syst. Man Cybernet.* **SMC-13** 834–46

Sugeno M (ed) 1985 *Industrial Applications of Fuzzy Control* (Amsterdam: Elsevier)

D3.1 Introduction

Toshihide Ibaraki

Abstract

See the abstract for Chapter D3.

If we view *evolutionary computation* (EC) as a means to find good suboptimal solutions of a given optimization problem, it is natural to consider its hybridization with existing optimization methods to improve its performance by exploiting their advantages. Such optimization methods range from exact algorithms studied in mathematical programming, such as integer programming, dynamic programming, branch and bound, polyhedral approaches, and linear and nonlinear programming, to heuristic (or approximate) algorithms tailored to given problem domains, such as greedy methods, local search (or hill climbing) and other heuristic constructions of solutions. The so-called metaheuristic algorithms, such as simulated annealing and tabu search, also aim at a similar goal, and can be combined with EC, even if they are at the same time tough competitors to EC.

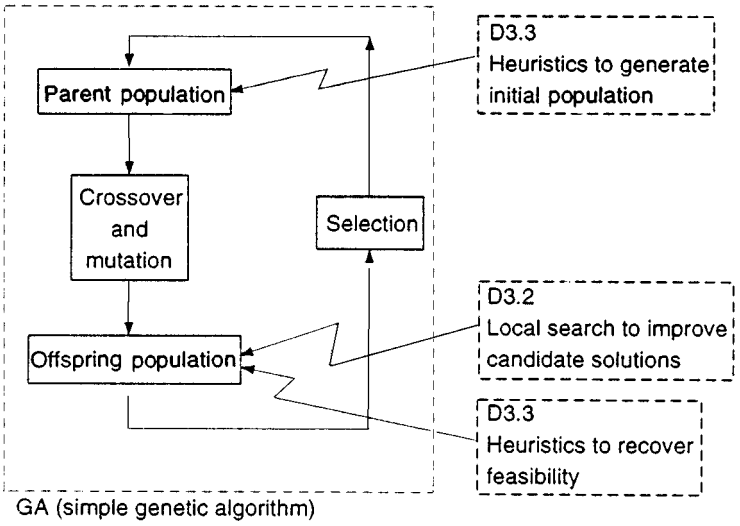


Figure D3.1.1. The genetic algorithm and its hybridization.

This chapter describes various possibilities of combining EC with such optimization methods, putting emphasis on combinatorial optimization problems. Perhaps a most natural and frequently attempted combination is that with the local search method; the resulting algorithm is often called *genetic local search*. Combinations with other methods, such as greedy methods, heuristic constructions of feasible solutions, dynamic programming, simulated annealing, and tabu search will also be described in some depth. Computational results of the resulting hybrid algorithms are reported to compare their performance with that of existing methods.

Throughout this chapter, we use a *simple genetic algorithm* as illustrated in figure D3.1.1, which B1.2 will be denoted as GA throughout this chapter, as a starting point of our discussion, rather than using the general framework of EC. This is mainly for the sake of simplicity, and most of the argument in this chapter

can be directly generalized to EC in an obvious manner. Furthermore, to make explanation more direct and understandable, we shall explain algorithms in terms of some combinatorial optimization problems, instead of presenting abstract and general description. In other words, we shall be mostly interested in the phenotype side of GA rather than the genotype, and describe how *solutions* of the problem under consideration are generated, modified, and selected, without referring to their genotype representation.

D3.2 Combination with local search

Toshihide Ibaraki

Abstract

See the abstract for Chapter D3.

D3.2.1 Introduction

Improving the performance of the genetic algorithm (GA) by utilizing the power of local search (or hill climbing) has been conceived since the very beginning of GA. Early literature such as the work of Brady (1985), Goldberg (1989), Davis (1991), Michalewicz (1992), Mühlenbein *et al* (1988), Mühlenbein (1989), Suh and van Gucht (1987), Jog *et al* (1991) (and possibly many others) has already mentioned the idea.

D3.2.2 Combinatorial optimization problems

Recall that an optimization problem requests a (globally) *optimal solution* x that minimizes its *objective function* (or *fitness function*) $f(x)$ among all *feasible solutions* $x \in S$, where S denotes the feasible region. Throughout this chapter, we shall consider minimization problems, unless otherwise stated. This does not lose generality because maximization of f is equivalent to minimization of $g = -f$. In many problems of interest, S and f are combinatorial in nature, and such problems are called *combinatorial optimization problems*.

D3.2.3 Neighborhood

Given a solution x, let $N(x)$ denote its *neighborhood*. Formally, $N(x)$ can be any set of solutions, but in most cases it is defined as a set of solutions obtained from x by perturbing its components in some specified manner. A feasible solution is *locally optimal* if there is no solution $y \in N(x) \cap S$ such that $f(y) < f(x)$.

As an example, consider the *traveling salesman problem* (TSP), which, given n points and distances between them, requests a shortest tour that visits every point exactly once before coming back to the initial point (see e.g. Lawler *et al* 1985). Among typical neighborhoods used for TSP, we mention here the p-opt neighborhood (Lin 1965), Or-opt neighborhood (Or 1976), and LK neighborhood (Lin and Kernighan 1973). G9.5

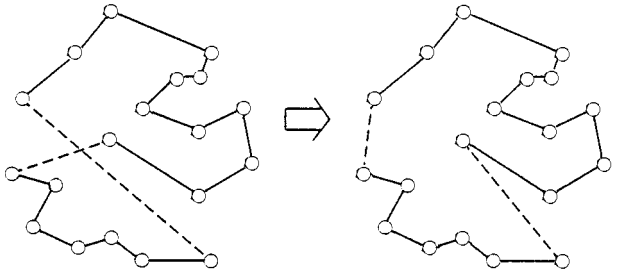
Given a tour $x = \{(i_1, i_2), (i_2, i_3), \ldots, (i_{n-1}, i_n), (i_n, i_1)\}$, where (i_k, i_{k+1}) denotes the edge connecting the kth point i_k and the $(k+1)$th point i_{k+1} in the tour, the p-opt neighborhood is defined by

$$N_p(x) = \{x' \mid x' \text{ is a tour obtained from } x \text{ by removing } p \text{ edges and adding the same number of appropriate edges}\}.$$

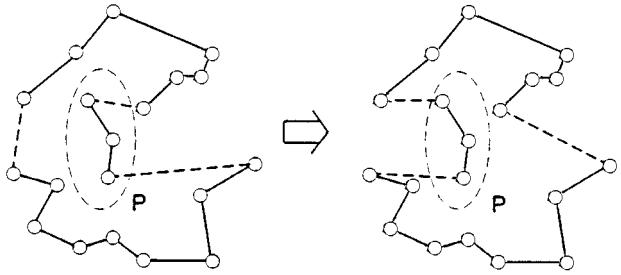
Here the *appropriate* edges means that the resulting set of edges again represents a tour (see figure D3.2.1(a) for a 2-opt neighbor). Removal of all possible p edges and addition of all possible appropriate edges are considered in this definition. In practice, $p \in \{2, 3\}$ are used, for which $|N_2(x)| = O(n^2)$ and $|N_3(x)| = O(n^3)$ hold.

The Or-opt neighborhood $N_{OR}(x)$ is a subset of $N_3(x)$ consisting only of those tours obtainable from x by removing a subpath P of at most three points and inserting it in another part of the tour (see

figure D3.2.1(b)). The LK neighborhood $N_{\mathrm{LK}}(x)$ is more sophisticated, and is defined by considering progressively larger p of $N_p(x)$, while restricting the candidates only to the promising ones chosen by certain heuristic criteria (see the article by Lin and Kernighan (1973) for details).



(a) A 2-opt neighbor



(b) An Or-opt neighbor

Figure D3.2.1. An illustration of 2-opt and Or-opt neighbors.

D3.2.4 Local search

Assume that neighborhood $N(x)$ is defined for any feasible solution x of the problem with feasible region S and objective function f. The following *local search* (LS) procedure can then be applied to improve x to a locally optimal solution:

Algorithm LS (*for minimization*)
Input: A feasible solution x.
Output: A locally optimal solution x'.
Step 0 (*initialization*): $k := 1$ and go to step k.
Step k (*improvement*): If there is a solution $y \in N(x) \cap S$ such that $f(y) < f(x)$, then $x := y, k := k + 1$ and return to step k. Otherwise, output x as x', and halt.

There are various implementation issues of LS:

- whether solutions in $N(x)$ are searched randomly or systematically
- whether a first improved solution found in $N(x) \cap S$ is immediately used for the next iteration (this strategy is called FIRST) or the best one in $N(x) \cap S$ is used (this is called BEST).

The solution x' obtained by LS is locally optimal with respect to neighborhood N. Although there is no guarantee that the locally optimal solution is also globally optimal, LS has been successfully applied to many problems to yield good suboptimal solutions.

D3.2.5 Handling infeasible solutions

Another important issue with LS is how to deal with infeasible (i.e. lethal) solutions in $N(x)$. For problems with very small $N(x) \cap S$ (or empty in some cases), it may be more advantageous to accept infeasible

solutions x as well, by considering the modified objective function f' with the *penalty to infeasibility* p added:

$$f'(x) = f(x) + Ap(x)$$

where $p(x) > 0$ if x is infeasible and $p(x) = 0$ otherwise, and $A \geq 0$ is a nonnegative weight. LS is then executed with $N(x)$, instead of $N(x) \cap S$. If weight A is appropriately controlled, there is a very good chance that the minimization of f' will eventually lead to an optimal solution of the original problem. (See Chapter C5 and Section D3.3.4 for other ways of handling infeasibility.) C5, D3.3.4

D3.2.6 Multistart local search and greedy randomized adaptive search procedure

It is natural to repeat the above LS many times from different initial solutions, which are usually generated randomly, until a certain termination criterion holds (e.g. the preassigned time bound has been consumed, or no improvement has been attained in a specified number of recent iterations). This is called *multistart local search* (MLS).

The *greedy randomized adaptive search procedure* (GRASP) (see e.g. Laguna *et al* 1994) is a variant of MLS, in which the initial solutions are generated by randomized greedy heuristics (see Section D3.3.3). D3.3.3 This is based on the idea that better initial solutions will lead to better locally optimal solutions.

D3.2.7 Genetic local search

As it is expected that GA can capture a global view of the entire solution space from the population of solutions at hand, the addition of the sharp optimization power of LS may benefit both approaches. The incorporation of LS into GA is usually performed as follows (see figure D3.1.1):

> In each generation of GA, apply the LS operator to all solutions in the offspring population, before applying the selection operator.

The resulting algorithm is generally called *genetic local search* (GLS). An alternative method is to apply LS to the parent population (i.e. after the selection is made) instead of the offspring population. Implementation details of LS, as discussed after the description of the LS algorithm, also have great influence on the performance of the resulting GLS.

D3.2.8 Computational results with genetic local search

GLS has been implemented for many combinatorial optimization problems. Taking TSP as an example, attempts with 2-opt, Or-opt and LK neighborhoods were made by, for example, Mühlenbein *et al* (1988), Jog *et al* (1991), Ulder *et al* (1991), and Kolen and Pesch (1994). The reported results are competitive with other existing heuristic algorithms for TSP. It is observed that more powerful local search (i.e. larger neighborhoods or more LS iterations) tends to give solutions of higher quality at the cost of consuming more computation time, suggesting that there is an appropriate tradeoff between GA and LS.

Table D3.2.1 is an excerpt from the article by Ulder *et al* (1991), in which GLS with 2-opt and LK neighborhoods is tested, together with MLS with the same neighborhoods and simulated annealing (SA) (see Section D3.5.2 for a description) with a 2-opt neighborhood. The crossover operator employed here D3.5.2 is the one proposed by Mühlenbein *et al* (1988). It shows the average relative errors (as percentages) from the optimum values for eight TSP instances taken from the literature, when all algorithms are run for the same amount of time $\bar{t}$ (in seconds), indicated in the second column. We may observe that the choice of neighborhood has a strong influence on performance: LK gives much better results than 2-opt. Also, GLS is more effective than MLS in improving overall performance. SA with 2-opt is also competitive; but the LK neighborhood is not compatible with SA since the random choice of SA is difficult to conduct in the LK neighborhood. From these, we may conclude that combination of GA and LS is a good strategy, which can yield better performance than that of GA or LS alone.

A comprehensive comparison among various algorithms is also given in Section D3.6.1 for the single D3.6.1 machine scheduling problem.

Table D3.2.1. A performance comparison of LS-based algorithms for the TSP. (Adapted from Ulder *et al* (1991).)

Instances	$\bar{t}$	SA (2-opt)	MLS (2-opt)	MLS (LK)	GLS (2-opt)	GLS (LK)
GRO48	6	1.89	1.35	0	0.19	0
TOM57	10	1.94	1.34	0	0.50	0
EUR100	60	2.59	3.23	0	1.15	0
GRO120	86	2.94	4.57	0.08	1.42	0.05
LIN318	1 600	2.37	6.35	0.37	2.02	0.13
GRO442	4 100	2.60	9.29	0.27	3.02	0.19
GRO532	8 600	2.77	8.34	0.37	2.99	0.17
GRO666	17 000	2.19	8.67	1.18	3.45	0.36

D3.2.9 Darwinian versus Lamarckian approaches

There are two types of approach to the generation of offspring in the above GLS algorithm, reflecting the two *evolutionary theories* of Darwin and Lamarck. Darwin says that the genetic code of an individual is inborn and never changes in its life. If we take this view, the crossover and/or mutation operations should be applied to the solutions *before* the improvement by LS is performed. On the other hand, Lamarck asserts that the genetic code changes during its life. In this viewpoint, the crossover and/or mutation should be applied to the solutions *after* LS is performed. As it is reported that the Lamarckian approach is more efficient (see e.g. Ackley and Littman 1994, Grefenstette 1991, Renders and Flasse 1996), the description in this chapter is based on the latter principle. A2.1

D3.2.10 Incorporation of other optimization tools

Optimization algorithms other than LS can also be incorporated to improve the solutions in the offspring or parent population of GA. For example, Kido (1993) uses SA in this way for the TSP. Powell *et al* (1989) are more flexible, and suggest using expert systems, which provide problem-specific knowledge, or simulation techniques, for this purpose. Other tools found in mathematical programming may also be employed. It is important, however, to keep the balance between the GA part and other optimization part, which can perhaps be achieved only via comprehensive computational experiment.

D3.2.11 Between genetic local search and multistart local search

There are different types of combination of GA and LS, which are located between GLS and MLS. The *iterated local search* (ILS) proposed by Johnson (1990) to solve the TSP by the LK neighborhood is known to be one of the most successful heuristic algorithms for the TSP. It operates in the following manner:

LS, mutation, LS, mutation, ...

until some termination criterion is satisfied. During all iterations, only a single solution is maintained, and the mutation is used to generate the next initial solution from the current locally optimal solution.

Another variant, discussed by Boese *et al* (1994) is between GLS and ILS, as it keeps a population of locally optimal solutions, but generates new initial solutions by a different method that somehow takes into account the effect of the whole population.

D3.2.12 Other optimization problems

The scope of genetic and GLS algorithms is not limited to combinatorial optimization problems. For example, continuous optimization problems, particularly global optimization of those problems with many local optimal solutions, have also been studied in some literature including the work of Michalewicz (1992) and Renders and Flasse (1996). It is reported that genetic operators such as crossover and mutation are useful to improve its reliability of finding global optimal solutions, while LS (e.g. the Newton method, quasi-Newton method, and simplex method) greatly improves the accuracy of the solutions obtained.

References

Ackley D H and Littman M L 1994 A case for Lamarckian evolution *Artificial Life III* ed C G Langton (Redwood City, CA: Santa Fe Institute) pp 3–10

Boese K D, Kahng A B and Muddu S 1994 A new adaptive multi-start technique for combinatorial global optimizations *Operations Res. Lett.* **16** 101–3

Brady R M 1985 Optimization strategies gleaned from biological evolution *Nature* **317** 804–6

Davis L (ed) 1991 *Handbook of Genetic Algorithms* (New York: Van Nostrand Reinhold)

Goldberg D E 1989 *Genetic Algorithms in Search, Optimization and Machine Learning* (Reading MA: Addison-Wesley)

Grefenstette J J 1991 Lamarckian learning in multi-agent environments *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, 1991)* ed R K Belew and L B Booker (San Mateo, CA: Morgan Kaufmann) pp 303–10

Jog P, Suh J Y and van Gucht D 1991 Parallel genetic algorithms applied to the traveling salesman problem *SIAM J. Optimization* **1** 515–29

Johnson D S 1990 Local optimization and the traveling salesman problem *Proc. 17th Colloq. on Automata, Languages and Programming* (New York: Springer) pp 446–61

Kido T 1993 Hybrid searches with genetic algorithms *Genetic Algorithms* ed H Kitano (Tokyo: Sangyo Tosho) pp 61–88 (in Japanese)

Kolen A and Pesch E 1994 Genetic local search in combinatorial optimization *Discrete Appl. Math.* **48** 273–84

Laguna M, Feo T A and Elrod H C 1994 A greedy randomized adaptive search procedure for the two-partition problem *Operations Res.* **42** 677–87

Lawler E L, Lenstra J K, Rinnoy Kan A H G and Shmoys D B 1985 *The Traveling Salesman Problem* (Chichester: Wiley)

Lin S 1965 Computer solutions of the traveling salesman problem *Bell Syst. Tech. J.* **44** 2245–69

Lin S and Kernighan B W 1973 An effective heuristic algorithm for the traveling salesman problem *Operations Res.* **21** 498–516

Michalewicz Z 1992 *Genetic Algorithms + Data Structures = Evolution Programs* (Berlin: Springer)

Mühlenbein H, Gorges-Schleuter M and Krämer O 1988 Evolution algorithms in combinatorial optimization *Parallel Comput.* **7** 65–85

Or I 1976 *Traveling Salesman-Type Combinatorial Problems and their Relation to the Logistics of Regional Blood Banking* PhD Thesis, Northwestern University

Powell D J, Tong S S and Skolnick M M 1989 EnGENEous domain independent, machine learning for design optimization *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 151–9

Renders J-M and Flasse S P 1996 Hybrid methods using genetic algorithms for global optimization *IEEE Trans. Syst. Man Cybernet.* B **SMC-26** 243–58

Suh J Y and van Gucht D 1987 Incorporating heuristic information into genetic search *Proc. 2nd Int. Conf. on Genetic Algorithms (Pittsburgh, PA, 1987)* ed J J Grefenstette (Hillsdale, NJ: Erlbaum) pp 100–7

Ulder N L J, Aarts E H L, Bandelt H-J, van Laarhoven P J M and Pesch E 1991 Genetic local search algorithms for the traveling salesman problem *Proc. 1st Conf. on Parallel Problem Solving from Nature (Dortmund, 1990)*) (*Lecture Notes in Computer Science 496*) ed H-P Schwefel and R Männer (Berlin: Springer) pp 109–16

D3.3 Uses of problem-specific heuristics

Toshihide Ibaraki

Abstract

See the abstract for Chapter D3.

D3.3.1 Introduction

As most of the interesting combinatorial optimization problems are intractable, as evidenced by the theoretical result of NP-hardness (Garey and Johnson 1979), efforts have been directed to develop efficient heuristic algorithms, which find good suboptimal solutions quickly. After explaining one such algorithm for the knapsack problem, we shall discuss how such heuristics can be utilized in the framework of the genetic algorithm (GA).

D3.3.2 The knapsack problem and a greedy heuristic

The *knapsack problem* (KP) is a 0-1 integer programming problem with a single inequality constraint: G9.7

$$\text{maximize } f(x) = \sum_{j=1}^{n} c_j x_j$$

$$\text{subject to } \sum_{j=1}^{n} a_j x_j \le b$$

$$x_j = 0 \text{ or } 1 \qquad j = 1, 2, \ldots, n$$

where a_j, c_j, and b are given positive integers. A *greedy heuristic* for this problem first arranges the indices j in the order

$$c_1/a_1 \ge c_2/a_2 \ge \ldots \ge c_n/a_n.$$

Then, starting with $b' := b$ and $f := 0$, it repeats the following operation for $j = 1, 2, \ldots, n$:

$$\text{if } a_j \le b', \text{ let } x_j := 1, b' := b' - a_j, f := f + c_j; \text{ otherwise let } x_j := 0. \tag{D3.3.1}$$

This algorithm always outputs a feasible solution, which is empirically known to be fairly good.

D3.3.3 Heuristics to generate the initial population

Simple greedy heuristics, as illustrated for the KP, can naturally be used to generate the initial population of GA (see figure D3.1.1), since the initial population consisting of good solutions is expected to facilitate convergence speed. For this purpose, however, it has to be modified so that a variety of solutions can be easily generated. A common gimmick is to introduce randomness by considering a candidate set at each iteration of the greedy heuristic algorithm. In the case of the above algorithm for the KP, a candidate set J of an appropriate size, containing indices j with large c_j/a_j among those x_j not fixed yet, is prepared and one index $j \in J$ is randomly chosen in each iteration of (D3.3.1). This type of modification is called a *randomized greedy heuristic*. As each run usually generates a different solution, a set of initial solutions can be prepared by repeating such an algorithm a certain number of times.

D3.3.4 Heuristics to recover feasibility

Another use of heuristics is to recover feasibility of those solutions generated by crossover and/or mutation in GA. For example, if a solution y of the 0–1 vector for the KP is not feasible (i.e. violates constraint $\sum_{j=1}^{n} a_j y_j \leq b$), we can apply operation (D3.3.1) only to those j satisfying $y_j = 1$. In this way, a feasible solution x can be obtained by modifying the solution y.

For some problems, crossover and mutation operators may generate many infeasible (i.e. lethal) solutions, but, if these infeasible solutions are modified into feasible solutions by using heuristics in the above manner, GA can avoid the danger of a premature convergence that occurs when all or most solutions are lethal (i.e. the effective population is very small).

D3.3.5 An example from the job-shop scheduling problem

A successful usage of heuristics of the above type can be found in the *job-shop scheduling problem* (JOB-SHOP). It consists of n jobs $J_1, J_2, \ldots, J_n$ and m machines $M_1, M_2, \ldots, M_m$. Each job J_j has an ordered list of m tasks $L_j = (T_{ji_1}, T_{ji_2}, \ldots, T_{ji_m})$ and processing times $p(T_{ji_k})$ of such tasks, meaning that these tasks must be processed in the order L_j, and processing of each task T_{ji_k} must be done on machine M_{i_k} consuming $p(T_{ji_k})$ time. Each machine can process at most one task at a time, and processing of a task cannot be interrupted (i.e. no preemption is allowed). JOB-SHOP requests an optimal schedule that minimizes its makespan (i.e. the time to complete all tasks). F1.5, G9.4

As an example, consider a JOB-SHOP instance specified by

$$L_1 = (T_{11}\ (5), T_{12}\ (4), T_{13}\ (3))$$
$$L_2 = (T_{23}\ (5), T_{21}\ (2), T_{22}\ (5))$$
$$L_3 = (T_{32}\ (3), T_{31}\ (6), T_{33}\ (2))$$

where the numbers in parentheses give processing times. A schedule for this instance is illustrated in the Gantt chart of figure D3.3.1. In this figure, the time to process a task T is represented by a bar of length $p(T)$, and we see that the makespan of this schedule is 15. Note that the sequence of bars in the line of machine M_i specifies the order of tasks on M_i. However, if an arbitrary order of tasks is given for each M_i, it may not yield a feasible schedule because a closed cycle may be created from the two types of order on M_i and on L_j. To resolve this difficulty, some heuristic algorithms have been proposed (see e.g. Giffler and Thompson 1969, Barker and McMahon 1985), which can repair the infeasibility in the given specification and construct a good feasible schedule.

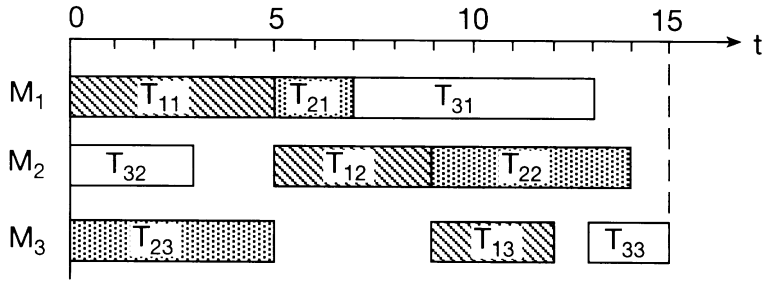


Figure D3.3.1. A Gantt chart of a schedule for the above JOB-SHOP instance.

Even though JOB-SHOP is known to be a very hard combinatorial problem, some GAs combined with heuristics to recover feasibility proved to be quite successful (see e.g. Yamada and Nakano 1992, Kobayashi *et al* 1993), and, for example, could find an optimal solution of the $n \times m = 10 \times 10$ instance in the book by Muth and Thompson (1963).

References

Barker J R and McMahon G B 1985 Scheduling the general job-shop *Management Sci.* **31** 594–98

Garey M R and Johnson D S 1979 *Computers and Intractability: a Guide to the Theory of NP-Completeness* (New York: Freeman)

Giffler B and Thompson G L 1969 Algorithms for solving production scheduling problems *Operations Res.* **8** 487–503
Kobayashi S, Ono I and Yamamura M 1993 Performance analysis of genetic algorithms for job-shop scheduling problems *Production Scheduling Symp.* pp 27–32 (in Japanese)
Muth J F and Thompson G L 1963 *Industrial Scheduling* (Englewood Cliffs, NJ: Prentice-Hall)
Yamada T and Nakano R 1992 *Genetic Algorithms for the Job-Shop Scheduling Problem* Technical Report of the Information Processing Society of Japan AI-8, pp 1–10 (in Japanese)

D3.4 Combination with dynamic programming

Toshihide Ibaraki

Abstract

See the abstract for Chapter D3.

D3.4.1 Introduction

A classical optimization method, dynamic programming (DP) can be used to find a best solution obtainable from a crossover of two solutions. We explain this by using the *single-machine scheduling problem* (SMP) as an example. Similar ideas can also be applied to other problems in which optimal sequences are sought. F1.5

D3.4.2 The single-machine scheduling problem

The SMP requests the determination of an optimal sequence of n jobs $J_1, J_2, \ldots, J_n$ to be processed on a single machine M without idle time. A sequence $\sigma = (\sigma(1), \sigma(2), \ldots, \sigma(n))$ is a permutation of $V = \{1, 2, \ldots, n\}$, where $\sigma(k) = j$ denotes that the kth job processed on the machine is J_j (conversely, $\sigma^{-1}(j)$ denotes the location of J_j in the sequence). Each job J_j requires processing time p_j and incurs cost $g_j(c_j)$ if completed at time c_j, where $c_j = \sum_{i=1}^{\sigma^{-1}(j)} p_{\sigma(i)}$. A sequence σ is optimal if it minimizes

$$\text{cost}(\sigma) = \sum_{j=1}^{n} g_j(c_j). \tag{D3.4.1}$$

In particular, in the following computational experiment, we consider the cost function

$$g_j(c_j) = h_j \max\{d_j - c_j, 0\} + w_j \max\{0, c_j - d_j\}$$

where d_j is the due date of J_j, and $h_j, w_j \geq 0$ are the weights given to earliness and tardiness of J_j, respectively.

D3.4.3 Dynamic programming for the single-machine scheduling problem

The following DP recursion due to Held and Karp (1962) can solve the SMP exactly, where $f^*(U)$ for $U \subseteq V$ denotes the minimum of (D3.4.1) summed over U when all jobs J_j, $j \in U$, are sequenced in the first $|U|$ positions.

$$\begin{aligned} f^*(\emptyset) &= 0 \\ f^*(U) &= \min_{j \in U} \left\{ f^*(U - \{j\}) + g_j\Big(\sum_{i \in U} p_i\Big) \right\} \qquad (\emptyset \neq)\ U \subseteq V. \end{aligned} \tag{D3.4.2}$$

Then $f^*(V)$ gives the cost of an optimal sequence of all jobs. The computation can be carried out in the nondecreasing order of $|U|$ in $O(n2^n)$ time, which however is not practical unless, for example, $n \leq 20$.

Now, given two sequences σ_1 and σ_2, let D denote the partial order such that $(i, j) \in D$ holds if and only if J_i is processed before J_j in both σ_1 and σ_2. Then the $f_D^*(V)$ computed by the following recursion

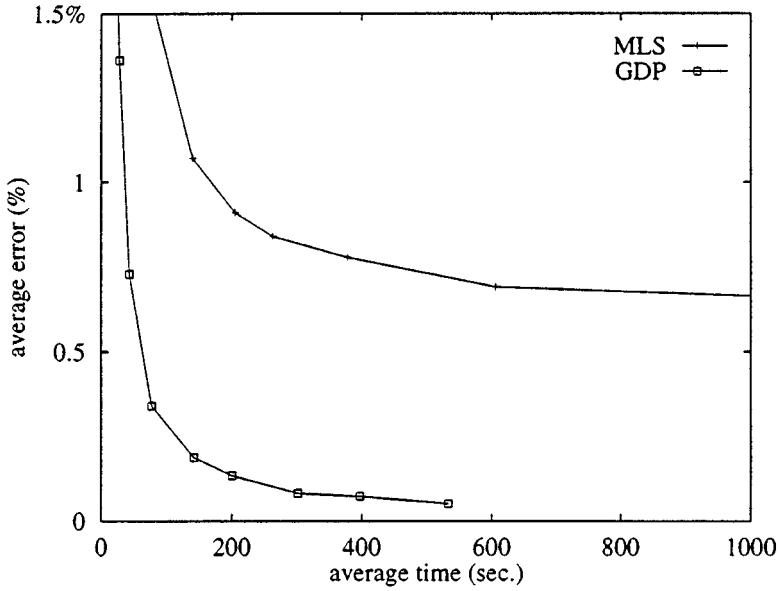


Figure D3.4.1. A comparison of GDP with MLS for the SMP.

gives the minimum cost over those sequences σ satisfying $\sigma^{-1}(i) < \sigma^{-1}(j)$ for all $(i, j) \in D$ (Yagiura and Ibaraki 1996).

$$f_D^*(\emptyset) = 0$$
$$f_D^*(U) = \min_{j \in I(U)} \left\{ f_D^*(U - \{j\}) + g_j\Big(\sum_{i \in U} p_i\Big) \right\} \qquad (\emptyset \neq)\ U \subseteq V^*(D) \tag{D3.4.3}$$

where

$$V^*(D) = \{U \subseteq V \mid j \in U \text{ and } (i, j) \in D \text{ imply } i \in U\}$$
$$I(U) = \{i \in U \mid \text{no } j \in U \text{ satisfies } j \neq i \text{ and } (i, j) \in D\}.$$

Solving this recursion may be regarded as the crossover of two solutions σ_1 and σ_2 (followed by local optimization). Therefore algorithm GA of figure D3.1.1 can be constructed by this crossover operator; it is called *genetic* DP (GDP). In the actual implementation, a mechanism is added to prevent spending too much time on solving (D3.4.3) (which can occur if D is not tight) by restricting the search space to only a subset of possible solutions.

Figure D3.4.1 compares the performance of GDP with that of multistart local search (MLS) (see Section D3.2.6) with the shift neighborhood D3.2.6

$$N(\sigma) = \{\sigma' \mid \sigma' \text{ is obtained from } \sigma \text{ by a shift operation } i \to j \text{ for some } i, j \in V\}$$

where $i \to j$ means that $\sigma(i)$ is moved to the location between $\sigma(j-1)$ and $\sigma(j)$. The figure shows how the average relative errors (as percentages) from the best known solutions improve as the computation time increases, where test instances are randomly generated for $n = 100$. A clear superiority of GDP over MLS may be concluded.

References

Held M and Karp R M 1962 A dynamic programming approach to sequencing problems *SIAM J Appl. Math.* **10** 196–210

Yagiura M and Ibaraki T 1996 The use of dynamic programming in genetic algorithms for permutation problems *Eur. J. Operational Res.* **92** 387–401

D3.5 Simulated annealing and tabu search

Toshihide Ibaraki

Abstract

See the abstract for Chapter D3.

D3.5.1 Introduction

In this section, we describe simulated annealing and tabu search, as these are also very effective methods discussed in metaheuristics, and then consider how to combine them with the genetic algorithm (GA).

D3.5.2 Simulated annealing

Let us first describe the algorithm of simulated annealing (SA; Kirkpatrick *et al* 1983) for an optimization problem with objective function f and feasible region S (see Section D3.2.2). In the algorithm, $N(x)$ denotes the neighborhood of a solution x (see Section D3.2.3) and z keeps the best solution found so far. D3.2.2 D3.2.3

Algorithm SA (*for minimization*)
Step 0 (*initialization*): Fix parameters t (initial temperature), L (the number of inner loop iterations), and γ ($0 < \gamma < 1$; reduction rate of temperature). Find an initial feasible solution x, and let $z := x$ and $k := 1$. Go to step k.
Step k (*random local search*):
(1: *inner loop*) Repeat the following (a) and (b) L times.

(a) Find a $y \in N(x) \cap S$ randomly.
(b) Let $\Delta := f(y) - f(x)$. If $\Delta \leq 0$, then $x := y$, and let $z := y$ if $f(y) < f(z)$; otherwise let $x := y$ with probability $e^{-\Delta/t}$.

(2: *outer loop*) If a termination criterion is satisfied, output z and halt. Otherwise, let $t := \gamma t$, $k := k + 1$ and return to step k.

Examples of termination criteria are to halt if temperature is frozen (i.e. $t \leq t_0$, for a prespecified value t_0), if k exceeds a given bound, and if a given computation time has been exhausted. SA is different from the local search (LS) of Section D3.2.4 in that a solution $y \in N(x) \cap S$ is probabilistically accepted even if $f(y) > f(x)$ (i.e. the quality of solution degrades). The acceptance probability is higher if temperature t is higher, which is controlled by the cooling scheme specified by γ. D3.2.4

D3.5.3 Unifying simulated annealing and the genetic algorithm

Two algorithms SA and GA (in particular genetic local search (GLS)) share many common features but differ in the following respects.

- SA maintains only a single candidate solution x, while GA maintains a population of solutions
- GLS uses crossover and mutation operators to generate new solutions, in addition to neighborhood search
- SA selects a new worse solution with probability $e^{-\Delta/t}$, while GA has its own selection rule such as the *roulette wheel* selection (see e.g. Goldberg 1989). C2.2

Therefore, a general framework that includes SA and GLS as special cases is possible if we allow the maintenance a population of solutions, to generate new solutions by crossover, mutation, and neighborhood search, and to perform selection by the schemes of SA and GA (see e.g. Mahfoud and Goldberg 1992, Chen and Flann 1994). Chen and Flann (1994) consider 14 different algorithms resulting from such a general framework, and point out that the algorithm, which employs crossover and mutation for generating new solutions, and the selection scheme of SA, performs quite well for various test beds, including TSP. They attribute this success to the high quality of the solutions generated by crossover and mutation, and to the power of avoiding premature convergence by the selection scheme of SA.

D3.5.4 Tabu search

Tabu search (TS) is also based on local search, but differs from SA and MLS as it always moves to a best solution $y \in (N(x) \cap S) \setminus T$ even if y has a worse objective value than x (Glover 1989). Here T is a set of solutions defined by a *tabu list* (or short-term memory), which is introduced to avoid a cycling over a small number of solutions. A tabu list is for example implemented by

- storing a fixed number of solutions recently searched or
- storing a fixed number of moves made in the recent search

where a move refers to the change (e.g. changing of x_j from zero to one) made to generate a new solution from the current solution. The central part of TS is described as follows.

Algorithm TS (*for minimization*)
Step 0 (*initialization*): Find an initial feasible solution x, and let $z := x$, $T := \emptyset$ and $k := 1$. Go to step k.
Step k (*move*):
(1: *inner loop*) If $(N(x) \cap S) \setminus T = \emptyset$, go to (2). Otherwise find a best solution $y \in (N(x) \cap S) \setminus T$, and let $x := y$. If this y is better than z, then $z := y$. Return to step k after updating T and letting $k := k + 1$.
(2: *outer loop*) If a termination criterion is satisfied, output z and halt. Otherwise, modify T and return to step k, after letting $k := k + 1$.

Upon completing the inner loop, set T is strategically modified in step k(2) so that the search can explore a new region not visited so far. This is called *diversification* or *strategic oscillation.* In order to identify such an unvisited region, TS is usually equipped with a long-term memory, which keeps record of, e.g. the counts of variables being changed in the past search, and/or the duration of the variables being fixed to certain values.

As the basic iteration in the inner loop is deterministic, it is hard to unify TS and GA in the same manner as done for SA and GA. However, the idea of diversification in the outer loop has much in common with the idea of GA to give the population large variety. In fact, the relinking operation conceived for diversification in TS (Glover 1995), which interpolates and/or extrapolates the set of elite solutions obtained in the past search, is very close to the idea of crossover in GA. Also, it is suggested to maintain a set of solutions, and to apply TS to such solutions in parallel. Much has to be done in future research, however, regarding the possibility of combining TS and GA.

References

Chen H and Flann N S 1994 Parallel simulated annealing and genetic algorithms: a space of hybrid methods *Parallel Problem Solving from Nature—PPSN III (Proc. Int. Conf. on Evolutionary Computation and 3rd Conf. on Parallel Problem Solving from Nature, Jerusalem, October 1994)* (*Lecture Notes in Computer Science 866*) ed Yu Davidor, H-P Schwefel and R Männer (Berlin: Springer) pp 428–38

Glover F 1989 Tabu search, part I *ORSA J. Comput.* **1** 190–206

——1995 *Tabu Search Fundamentals and Uses* Technical Report, University of Colorado

Goldberg D E 1989 *Genetic Algorithms in Search, Optimization and Machine Learning* (Reading MA: Addison-Wesley)

Kirkpatrick S, Gelatt C D and Vecchi M P 1983 Optimization by simulated annealing *Science* **220** 671–80

Mahfoud S W and Goldberg D E 1992 A genetic algorithm for parallel simulated annealing *Parallel Problem Solving from Nature, 2 (Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature, Brussels, 1992)* ed R Männer and B Manderick (Amsterdam: Elsevier) pp 301–10

D3.6 Comparison with existing optimization methods

Toshihide Ibaraki

Abstract

See the abstract for Chapter D3.

D3.6.1 Experimental results

To conclude this chapter, we cite the results from Yagiura and Ibaraki (1996), which compare the performance of the genetic algorithm (GA) and genetic local search (GLS) with other optimization methods such as multistart local search (MLS), greedy randomized adaptive search procedure (GRASP), simulated annealing (SA) and tabu search (TS), on a test bed problem of the single-machine scheduling problem (SMP; defined in Section D3.4.2). The components of these algorithms are determined as follows after some preparatory experiment. D3.4.2

- Neighborhood $N(\sigma)$ of the current solution σ consists of the solutions obtained by *swap* operations; that is, σ' is in $N(\sigma)$ if it is obtained from $\sigma = (\sigma(1), \sigma(2), \ldots, \sigma(n))$ by interchanging a pair of $\sigma(i)$ and $\sigma(j)$ for some $i \neq j$. Therefore, $|N(\sigma)| = \mathrm{O}(n^2)$.
- Initial solutions are always generated randomly, except that GRASP uses randomized greedy heuristics for this purpose.
- The local search operator in GLS, MLS, and GRASP is implemented by the FIRST strategy (see Section D3.2.4). D3.2.4
- The order crossover with uniform mask (Mühlenbein *et al* 1988, Davis 1991) and the mutation by random swap operations are used in GA and GLS.
- The size of population is 1000 for GA and 20 for GLS.
- The selection of solutions in GA and GLS is deterministically made by preferring those with smaller cost values under the constraint that all solutions are different.

The results are summarized in figure D3.6.1, which shows how the average relative errors (for 10 SMP instances with $n = 100$ jobs) decrease as the number of samples (i.e. the number of solutions whose cost values (D3.4.1) have been evaluated). The number of samples is roughly proportional to the required central processing unit (CPU) time.

The figure indicates that GLS is much more efficient than GA. Even with local search alone (i.e. MLS), it is more efficient than GA, and it can be further improved by employing good initial solutions (i.e. GRASP). GLS, GRASP, SA, and TS behave more or less similarly, when the number of samples exceeds 10^6; however GLS and SA perform slightly better than TS and GRASP. It should be noted that TS in this experiment is implemented without the diversification mechanism, and could be improved further (judging from the very good computational results of TS reported in the literature; see e.g. Glover *et al* 1993).

A comparison of various algorithms on JOB-SHOP (see Section D3.3.5) can be found in the articles by Aarts *et al* (1994) and Vaessens *et al* (1996). D3.3.5

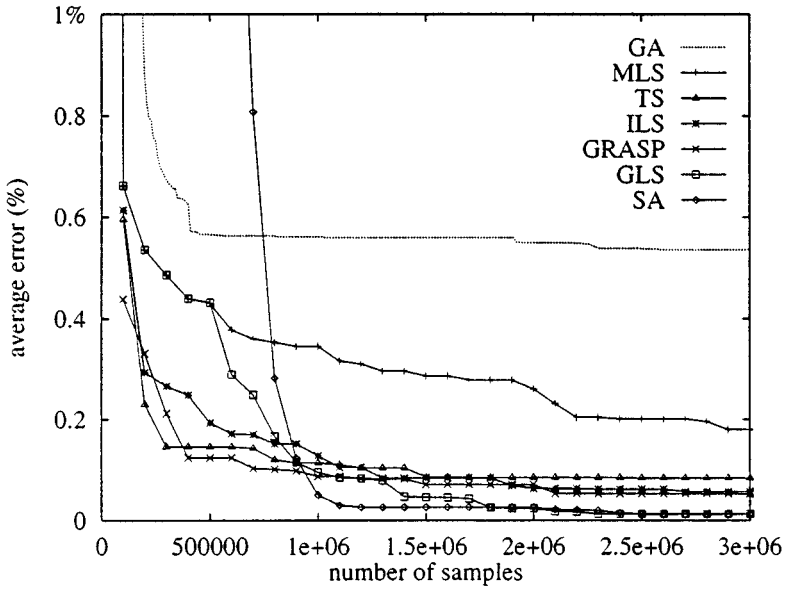


Figure D3.6.1. Performance of various optimization methods.

References

Aarts E H L, van Laarhoven P J M, Lenstra J K and Ulder N L J 1994 A computational study of local search algorithms for job-shop scheduling *ORSA J. Comput.* **6** 118–25

Davis L (ed) 1991 *Handbook of Genetic Algorithms* (New York: Van Nostrand Reinhold)

Glover F, Laguna M, Taillard E and de Werra D (eds) 1993 *Tabu Search* (Basel: Baltzer)

Mühlenbein H, Gorges-Schleuter M and Krämer O 1988 Evolution algorithms in combinatorial optimization *Parallel Comput.* **7** 65–85

Vaessens R J M, Aarts E H L and Lenstra J K 1996 Job shop scheduling by local search *INFORMS J. on Computing* **8** 302–17

Yagiura M and Ibaraki T 1996 Metaheuristics as robust and simple optimization tools *Proc. 1996 IEEE Int. Conf. on Evolutionary Computation* (Piscataway, NJ: IEEE) pp 541–6

E1.1 Population size

Robert E Smith

Abstract

This section examines a critical issue in the design of any evolutionary computation (EC) system: the sizing of the population. Although many empirical studies have considered the issue of population sizing, limited theoretical advice is available. Two primary theoretical arguments on population sizing are examined. The first is the maximization of schema processing rate through population sizing. The second is sizing of populations for appropriate schema sampling. Although both these theories are drawn from the genetic algorithm (GA) literature, they have broad applicability across a range of EC algorithms.

E1.1.1 Introduction

How large should a population be for a given problem? This question has been considered empirically in several studies (De Jong 1975, Grefenstette 1986, Schaffer *et al* 1989), and there are a variety of heuristic recommendations on sizing populations for a variety of EC algorithms. This section considers analytically motivated suggestions on population sizing. It is primarily focused on *genetic algorithms* (GAs), since most of the analytical work on sizing populations is in this EC subfield. However, many of the concepts discussed can be transferred to other EC algorithms. B1.2

The issue of population sizing can also be considered theoretically in the light of GA schema processing. By misinterpreting the *implicit parallelism* ($O(\mu^3)$) argument (see Section B2.5), one might conclude that the larger μ (the population size) is, the greater the computational leverage, and, therefore, the better the GA will perform. This is clearly not the case, since there are only 3^ℓ schemata in a binary-encoded GA with strings of length ℓ. One clearly cannot process $O(\mu^3)$ schemata if μ^3 is much greater than 3^ℓ. B2.5

There are several ways to view the sizing of a GA population. One is to size the population such that computational leverage (i.e. schema processing ability) is maximized. Goldberg (1989) shows how to set population size for an optimal balance of computational effort to computational leverage. This development is outlined below. However, it is important to note that there are other GA performance criteria that can (and should) be considered when sizing a population. One is the accuracy of schema average fitness values indicated by a finite sample of the schemata in a population. This issue will be considered later in this section. However, put in a broader context, population sizing cannot be considered in complete isolation from other GA parameters. Ultimately, the GA must balance computational leverage, accurate sampling of schemata, population diversity, mixing through recombination, and selective pressure for good performance.

E1.1.2 Sizing for optimal schema processing

To consider how the computational leverage of implicit parallelism can be maximized, one must thoroughly consider the number of schemata processed by the GA. One can derive an exact expected number of schemata in a population of size n given binary strings of length ℓ. Note that this argument can be extended to higher-cardinality alphabets as well.

First consider the probability that a single string matches a particular schema H:

$$P_H = \left(\tfrac{1}{2}\right)^{\mathrm{O}(H)}.$$

Given this, the probability of zero matches of this schema in a population of size μ is

$$\left[1 - \left(\tfrac{1}{2}\right)^{\mathrm{O}(H)}\right]^{\mu}.$$

Therefore, the probability of one or more matches in a population of size n is

$$1 - \left[1 - \left(\tfrac{1}{2}\right)^{\mathrm{O}(H)}\right]^{\mu}.$$

There are

$$\left(\begin{array}{c} \ell \\ \mathrm{O}(H) \end{array} \right) 2^{\mathrm{O}(H)}$$

schemata of order $\mathrm{O}(H)$ in a strings of length ℓ. Therefore, if one counts over all possible schemata, and considers the probability of one or more of those schemata in a population of size μ, the total, expected number of schemata in the population is

$$S(\mu, l) = \sum_{i=0}^{\ell} \left(\begin{array}{c} \ell \\ i \end{array} \right) 2^i \left\{ 1 - \left[1 - \left(\frac{1}{2} \right)^i \right]^{\mu} \right\}.$$

Consider schemata of defining length ℓ_{s} or less, such that these schemata are highly likely to survive crossover and mutation. These schemata can be thought of as building blocks. Given the previous count of schemata, one can slightly underestimate the number of building blocks as

$$n_{\mathrm{s}}(\mu, \ell, \ell_{\mathrm{s}}) = (\ell - \ell_{\mathrm{s}} + 1)\, S(\mu, \ell_{\mathrm{s}} - 1).$$

Note that the number of building blocks monotonically expands from $(\ell - \ell_{\mathrm{s}} + 1)\, 2^{\ell_{\mathrm{s}}}$ (for a population of one) to $(\ell - \ell_{\mathrm{s}} + 1)\, 3^{\ell_{\mathrm{s}}}$ (for an infinite population).

Given this count, a measure of the GA's computational leverage is $\mathrm{d}S/\mathrm{d}t$, the average real-time rate of schemata processing. Assume that the population ultimately converges to contain only one unique string, and, thus, 2^{ℓ} schemata. Therefore, for the overall GA run, one can estimate

$$\frac{\mathrm{d}S}{\mathrm{d}t} \approx \frac{(S_0 - 2^{\ell})}{\mathrm{D}t}$$

where S_0 is the expected number of unique schemata in the initial, random population (given by the S count equation above), and $\mathrm{D}t$ is the time to convergence.

Assume

$$\mathrm{D}t = n_{\mathrm{c}} t_{\mathrm{c}}$$

where n_{c} is the generations to convergence, and t_{c} is the real time per generation. Goldberg (1989) estimates the convergence time under fitness proportionate selection. If one considers convergence of all but λ percent of the population to one string, where λ is the initial percentage of the population occupied by the string, the time to convergence is constant with respect to population size. If one considers convergence to all but one of the population members to the same string, the convergence time is $n_{\mathrm{c}} = \mathrm{O}(\ln \mu)$.

The time t_{c} varies with the degree of parallelization, since parallel computers can evaluate several fitness values simultaneously. Therefore,

$$t_{\mathrm{c}} = \mu^{(1-\beta)}.$$

The value $\beta = 1$ represents a perfectly parallel computer, where all fitness values are evaluated at once. The value $\beta = 0$ represents a perfectly serial computer. Note that $\mathrm{D}t$ increases monotonically with n.

Since $\mathrm{D}S/\mathrm{D}t$ is given as an analytical function, one can find its maxima using standard numerical search techniques. Goldberg (1989) compiles maxima for several values of ℓ_{s} and β in plots and tables.

Surprisingly, this development for serial computers and the $\mathrm{O}(\ln \mu)$ convergence time assumption indicate that one should use the smallest population possible. A population of size three seems the smallest that is technically feasible, since two population members are required to be selected over the third, and then recombined to form a new population. If one starts with such small populations, convergence will

be rapid, and then the GA can be restarted. This inspired the *micro-GA* (Krishnakumar 1989), which uses very small populations, and repeated, partially random restarts. Although results of the micro-GA are promising, it is important to note that the optimal population size for schema processing rate *may not* be the optimal size for ultimate GA effectiveness. Sampling error may overwhelm the GA's ability to select correctly in small populations.

E1.1.3 Sizing for accurate schema sampling

Another study by Goldberg *et al* (1992) examines population sizing in terms of ultimate GA performance by considering sampling error. Basic GA theory suggests that GAs search by implicitly evaluating the mean fitness of various schemata based on a series of population samples, and then recombining highly fit schemata. Since the schema average fitness values are based on samples, they typically have a non-zero variance. Consider the competing schemata

$$H_1 = * \ * \ * \ * \ 1 \ 1 \ 0 \ * \ * \ 0$$
$$H_2 = * \ * \ * \ * \ 0 \ 1 \ 0 \ * \ * \ 0.$$

Assuming a deterministic fitness function, variance of average fitness values of these schemata exist due to the various combinations of bits that can be placed in the 'don't care' (*) positions. This variance has been called *collateral noise* (Goldberg and Rudnick 1991). Let $f(H_1)$ and $f(H_2)$ represent the average fitness values for schemata H_1 and H_2, respectively, taken over all possible strings in each schema. Also let σ_1^2 and σ_2^2 represent the variances taken over all corresponding schema members.

The GA does not make its selection decisions based on $f(H_1)$ and $f(H_2)$. Instead, it makes these decisions based on a sample of a given size for each schema. Let us call these observed fitness values $f_o(H_1)$ and $f_o(H_2)$. Observed fitness values are a function of $n(H_1)$ and $n(H_2)$, the numbers of copies of schemata H_1 and H_2 in the population, respectively. Given moderate sample sizes, the central-limit theorem† tells us that the f_o-values will be distributed normally, with mean $f(H)$ and variance $\sigma^2/n(H)$.

Due to the sampling process and the related variance, it is possible for the GA to err in its selection decisions on schema H_1 versus H_2. In other words, if one assumes $f(H_1) > f(H_2)$, there is a probability that $f_o(H_1) < f_o(H_2)$. If such mean fitness values are observed the GA may incorrectly select H_2 over H_1. Given the $f(H)$ and σ^2 values, one can calculate the probability of $f_o(H_1) < f_o(H_2)$ based on the convolution of the two normals. This convolution is itself normal with mean $f(H_1) - f(H_2)$ and variance $(\sigma_1^2/n(H_1)) + (\sigma_2^2/n(H_2))$. Thus, the probability that $f_o(H_1) < f_o(H_2)$ is α, where

$$z^2(\alpha) = \frac{(f(H_1) - f(H_2))^2}{(\sigma_1^2/n(H_1)) + (\sigma_2^2/n(H_2))}$$

and $z(\alpha)$ is the ordinate of the unit, one-sided, normal deviate. Note that $z(\alpha)$ is, in effect, a signal-to-noise ratio, where the signal in question is a selective advantage, and the noise is the collateral noise for the given schema competition.

For a given z, α can be found in standard tables, or approximated. For values of $|z| > 2$ (two standard deviations from the mean), one can use the Gaussian tail approximation:

$$\alpha = \frac{\exp\left(-z^2/2\right)}{\left(z(2\pi)^{1/2}\right)}.$$

For values of $|z| \leq 2$, one can use the sigmoidal approximation suggested by Valenzuela-Rendon (1989):

$$\alpha = \frac{1}{1 + \exp(-1.6z)}.$$

Given this calculation, one can match a desired maximum level of error in selection to a desired population size. This is accomplished by setting $n(H_1)$ and $n(H_2)$ such that the error probability is lowered

† Technically, the central-limit theorem only applies to a random sample. Therefore, the assumption that the mean of observed, average fitness values is an unbiased sample of the average fitness values over all strings is only valid in the initial, random population, and perhaps in other populations early in the GA run. However, GA theory makes the assumption that selection is sufficiently slow to allow for good schema sampling.

below the desired level. In effect, raising either of the $n(H)$ values 'sharpens' (lowers the variance of) the associated normal distribution, thus reducing the convolution of the two distributions.

Goldberg *et al* (1992) suggest that if the largest value of $2^{O(H)}\sigma_m^2/|f(H_1) - f(H_2)|$ (where σ_m is the mean schemata variance, $\left(\sigma(H_1)^2 + \sigma(H_2)^2)/2\right)$) is known for competitive schemata of order $O(H)$, one can conservatively size the population by assuming the $n(H)$ values are the expected values for a random population of size μ. This gives the sizing formula:

$$\mu = 2z^2(\alpha)2^{O(H)}\frac{\sigma_m^2}{(f(H_1) - f(H_2))^2}.$$

Note that this formula can be extended to alphabets of cardinality greater than two.

The formula is a thorough compilation of the concepts of schema variance and its relationship to population sizing. However, it does present some difficulties. The values and ranges of $f(H)$ are not known beforehand for any schemata, although these values are implicitly estimated in the GA process. Moreover, the values of σ^2 are neither known nor estimated in the usual GA process. Despite these limitations, Goldberg *et al* (1991) suggest some useful rules of thumb for population sizing from this relationship.

For instance, consider problems with deception of order $k \ll \ell$. That is, all building blocks of order k or less have no deception. One could view such a function as the sum of $m = \ell/k$ subfunctions, f_i. Thus, the root-mean-squared variance of a subfunction is

$$\sigma_{rms}^2 = \frac{\sum_{i=1}^{m}\sigma_{f_i}^2}{m}.$$

An estimate of the variance of the average order-k schema is

$$\sigma_m^2 = (m-1)\sigma_{rms}^2.$$

This gives a population of size

$$\mu = 2z^2(\alpha)2^k(m-1)\frac{\sigma_{rms}^2}{(f(H_1) - f(H_2))^2}.$$

Note that the population size is $O(m) = O(\ell/k) = O(\ell)$ for problems of fixed, bounded deception order k.

This relationship suggests the rule of thumb that an adequate population size increases linearly with string length for problems of fixed, bounded deception. Moreover, it has some interesting implications for GA time complexity. Goldberg and Deb (1990) show that for typical selection schemes GAs converge in $O(\log\mu)$ or $O(\mu\log\mu)$ generations. This suggests that GAs can converge in $O(\ell\log\ell)$ generations, even when populations are sized to control selection errors.

One can construct another rule of thumb by considering the maximum variance in a GA fitness function, which is given by

$$\sigma_f^2 = \frac{(f_{max} - f_{min})^2}{4}$$

where f_{max} is the maximum fitness value, and f_{min} is the minimum fitness value for the function. One could use this value as a conservative estimate of the schema average fitness variance (the collateral noise), and size the population accordingly.

The population sizing formula has also suggested a method of dynamically adjusting population size. In a recent study (Smith 1993a, b, Smith and Smuda 1995), a modified GA is suggested that adaptively resizes the population based on the absolute expected selection loss, which is given by

$$L(H_1, H_2) = |f(H_1) - f(H_2)|\alpha(H_1, H_2)$$

where α is derived from the previous formula, and competitions of mates that estimate not only schema average finesses (as in the usual GA), but schema fitness variances as well. Note that the $L(H_1, H_2)$ measure considers not only the variance of a schema competition, but also its relative effect. Note that this is important in an adaptive sizing algorithm, since the previous population sizing formula does not consider the relative importance of schemata competitions. If two competing schemata have fitness values

that are nearly equal, the overlap in the distributions will be great, thus suggesting a large population. However, if the fitness values of these schemata are nearly equal, their importance to the overall search may be minimal, thus precluding the need for a large population on their account.

Preliminary experiments with the adaptive population sizing technique have indicated its viability. They also suggest the possibility of other techniques that automatically and dynamically adjust population size in response to problem demands.

E1.1.4 Final comments

This section has presented arguments for sizing GA populations. However, the concepts (maximizing computational leverage and ensuring accurate sampling) are general, and can be applied to other EC techniques. In different situations, either of these two concepts may determine the best population size. In many practical situations, it will be difficult to determine which concept dominates. Moreover, population size based on these concepts must be considered in the context of recombinative mixing, disruption, deception, population diversity, and selective pressure (Goldberg *et al* 1993). One must also consider the implementation details of a GA on parallel computers. Specifically, how does one distribute subpopulations on processors, and how does one exchange population members between processors? Some of these issues are considered in recent studies (Goldberg *et al* 1995). As EC methods advance, automatic balancing of these effects based on theoretical considerations is a prime concern.

References

De Jong K A 1975 An analysis of the behavior of a class of genetic adaptive systems *Dissertation Abstracts Int.* **36** 5140B (University Microfilms No 76–9381)

Goldberg D E 1989 Sizing populations for serial and parallel genetic algorithms *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, June 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 70–9

Goldberg D E and Deb K 1990 *A Comparative Analysis of Selection Schemes used in Genetic Algorithms* TCGA Report 90007, The University of Alabama, The Clearinghouse for Genetic Algorithms

Goldberg D E, Deb K and Clark J H 1991 *Genetic Algorithms, Noise, and the Sizing of Populations* IlliGAL Technical Report 91010, University of Illinois at Urbana-Champaign

——1992 Accounting for noise in the sizing of populations *Foundations of Genetic Algorithms 2* ed L D Whitley (San Mateo, CA: Morgan Kaufmann) pp 127–40

Goldberg D E, Deb K and Thierens D 1993 Toward a better understanding of mixing in genetic algorithms *J. Soc. Instrum. Control Eng.* **32** 10–16

Goldberg D E, Kargupta H, Horn J and Cantu-Paz E 1995 *Critical Deme Size for Serial and Parallel Genetic Algorithms* IlliGAL Technical Report 95002, University of Illinois at Urbana-Champaign

Goldberg D E and Rudnick M 1991 Genetic algorithms and the variance of fitness *Complex Syst.* **5** 265–78

Grefenstette J J 1986 Optimization of control parameters for genetic algorithms *IEEE Trans. Syst. Man Cybernet.* **SMC-16** 122–8

Krishnakumar K 1989 Microgenetic algorithms for stationary and non-stationary function optimization *SPIE Proc. on Intelligent Control and Adaptive Systems* vol 1196 (Bellingham, WA: SPIE) pp 289–96

Schaffer J D, Caruana R A, Eshelman L J and Das R 1989 A study of control parameters affecting online performance of genetic algorithms for function optimization *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, June 1989)* (San Mateo, CA: Morgan Kaufmann) pp 51–60

Smith R E 1993a *Adaptively Resizing Populations: an Algorithm and Analysis* TCGA Report 93001, University of Alabama

——1993b Adaptively resizing populations: an algorithm and analysis *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* (San Mateo, CA: Morgan Kaufmann) p 653

Smith R E and Smuda E 1995 Adaptively resizing populations: algorithm, analysis, and first results *Complex Syst.* **9** 47–72

Valenzuela-Rendon M 1989 *Two Analysis Tools to Describe the Operation of Classifier Systems* TCGA Report 89005, The University of Alabama, The Clearinghouse for Genetic Algorithms

E1.2 Mutation parameters

Thomas Bäck

Abstract

In this section, heuristics for setting the mutation parameter values in evolutionary algorithms are discussed. Basically, mutation parameters that control the self-adaptation process in evolution strategies and evolutionary programming are distinguished from parameter settings affecting the mutation rate or mutation step size directly. The latter methods are often used in genetic algorithms and evolutionary heuristics derived from genetic algorithms. Commonly, such heuristics work with a constant setting of the mutation rate (as in genetic algorithms), but settings varying over time according to a deterministic or even a probabilistic schedule are also known and are summarized here.

E1.2.1 Introduction

The basic distinction between the concept of handling mutation in evolution strategies and evolutionary programming as opposed to genetic algorithms has already been clarified in Chapters B1 and C3: *evolution strategies* and *evolutionary programming* evolve their set of mutation parameters ($n_\sigma \in \{1, \ldots, n\}$ variances and $n_\alpha \in \{0, \ldots, (n - n_\sigma/2)(n_\sigma - 1)\}$ covariances of the generalized, n-dimensional normal distribution) on-line during the search by applying the search operator(s) mutation (and recombination, in case of evolution strategies) to the *strategy parameters* as well. This principle facilitates the *self-adaptation* of strategy parameters and shifts the parameter setting issue to the more robust level of the learning rates; that is, the parameters that control the speed of the adaptation of strategy parameters. Section E1.2.2 briefly discusses the presently used heuristics (which are based on some theoretical ground) for setting these learning rates on the meta-level of strategy parameter modifications.

B1.3 B1.4 C3.2.2, C7.1

In contrast to associating a (potentially large) number of mutation parameters with each single individual and self-adapting these parameters on-line, *genetic algorithms* and evolutionary heuristics derived from genetic algorithms usually provide only one *mutation rate* p_m for the complete population. This mutation rate is set to a fixed value, and it is not modified or self-adapted during evolution. A variety of values were proposed for setting p_m, and a summary of these results (which are obtained from experimental investigations) is given in section E1.2.3. In addition to constant settings of the mutation rate, some experiments with a mutation rate varying over the generation number are also provided in the literature, including efforts to calculate the optimal schedule of the mutation rate for simple objective functions and to derive some general heuristics from these investigations. Furthermore, the variation of the mutation rate might be probabilistic rather than deterministic, and p_m might also vary over bit representation in the case of *binary representation* of individuals. These mutation heuristics are discussed in section E1.2.3.

B1.2 C3.2.1 C1.2

E1.2.2 Mutation parameters for self-adaptation

In evolution strategies, the *mutation of standard deviations* σ_i ($i \in \{1, \ldots, n_\sigma\}$) according to the description given by Bäck and Schwefel (1993); that is,

C3.2.2

$$\sigma_i' = \sigma_i \exp(\tau' N(0, 1) + \tau N_i(0, 1)) \qquad \text{(E1.2.1)}$$

is controlled by two meta-parameters or learning rates τ and τ'. Schwefel (1977, p 167–8) suggests setting these parameters according to

$$\tau' = \frac{K}{(2n)^{1/2}} \qquad \text{and} \qquad \tau = \frac{K}{[2(2n)^{1/2}]^{1/2}} \tag{E1.2.2}$$

and recently Schwefel and Rudolph (1995) generalized this rule by setting

$$\tau' = \frac{K\delta}{(2n)^{1/2}} \qquad \text{and} \qquad \tau = \frac{K(1-\delta)}{[2n/(n_\sigma)^{1/2}]^{1/2}} \tag{E1.2.3}$$

where K denotes the (in general unknown) normalized *convergence velocity* of the algorithm. Although the convergence velocity K cannot be known for arbitrary problems, the parameters τ and τ' are very robust against settings deviating from the optimal value (a variation within one order of magnitude normally causes only a minor loss of efficiency). Consequently, a setting of $K = 1$ is a useful initial recommendation. B2.4

Experiments varying the weighting factor δ have not been performed so far, such that the default value $\delta = 1/2$ should be used for first experiments with an evolution strategy.

For the mutation of rotation angles α_j in evolution strategies with *correlated mutations*, which is performed according to the rule C3.2.2

$$\alpha_j' = \alpha_j + \beta N_j(0, 1) \tag{E1.2.4}$$

a value of $\beta = 0.0853$ (corresponding to 5°) is recommended on the basis of experimental results.

For the simple self-adaptation case $n_\sigma = 1$, where only one standard deviation is learned per individual, equation (E1.2.1) simplifies to $\sigma' = \sigma \exp(\tau_0 N(0, 1))$ with a setting of $\tau_0 = K/n^{1/2}$. Alternatively, Rechenberg (1994) favors a so-called *mutational step size* control which modifies σ according to the even simpler rule $\sigma' = \sigma u$, where $u \sim U(\{1, \alpha, 1/\alpha\})$ is a uniform random value attaining one of the values 1, α and $1/\alpha$. This is motivated by the idea of trying larger, smaller, and constant standard deviations in the next generation, each of these with one-third of the individuals. Rechenberg (1994, p 48) recommends a value of $\alpha = 1.3$ for the learning rate. An experimental comparison of both self-adaptation rules for $n_\sigma = 1$ has not been performed so far. C3.2.2

In case of evolutionary programming (EP), Fogel (1992) originally proposed an additive, normally distributed mutation of variances for self-adaptation in meta-EP, but subsequently substituted the same logarithmic-normally distributed modification as used in evolution strategies (Saravanan and Fogel 1994a, 1994b). Consequently, the parameter setting rules for τ' and τ also apply to evolutionary programming.

E1.2.3 Mutation parameters for direct schedules

Holland (1975) introduced the *mutation operator* of genetic algorithms as a 'background operator' that changes bits of the individuals only occasionally, with a rather small mutation probability $p_m \in [0, 1]$ per bit. Common settings of the mutation probability are summarized in table E1.2.1. C3.2.1

Table E1.2.1. Commonly used constant settings of the mutation rate p_m in genetic algorithms.

p_m	Reference
0.001	De Jong (1975, pp 67–71)
0.01	Grefenstette (1986)
0.005–0.01	Schaffer *et al* (1989)

These settings were all obtained by experimental investigations, including a meta-level optimization experiment performed by Grefenstette (1986), where the space of parameter values of a genetic algorithm was searched by another genetic algorithm. Mutation rates within the range of values summarized in table E1.2.1 are still widely used in applications of canonical (i.e. using binary representation) genetic algorithms, because these settings are consistent with Holland's proposal for mutation as a background operator and Goldberg's recommendation to invert on the order of one per thousand bits by mutation (Goldberg 1989, p 14). Although it is correct that *base pair mutations* of *eschericia coli* bacteria occur with a similar frequency (Futuyma 1990, pp 82–83), it is important to bear in mind that this reflects mutation rates in a relatively late stage of evolution and in only one specific example of natural evolution, A2.1

which may or may not be relevant to genetic algorithms. Early stages in the history of evolution on earth, however, were characterized by much larger mutation rates (see Ebeling *et al* 1990, ch 8).

Taking this into account, some authors proposed varying the mutation rate in genetic algorithms over the number of generations according to some specific, typically decreasing schedule—which is usually deterministic, but might also be probabilistic. Fogarty (1989) performed experiments comparing, for binary strings of length $\ell = 70$, the following schedules:

(i) a constant mutation rate $p_m = 0.01$

(ii) a mutation rate

$$p_m(t) = \frac{1}{240} + \frac{0.11375}{2^t} \tag{E1.2.5}$$

that is, a schedule where the mutation rate decreases exponentially over time

(iii) a mutation rate varying over bit representations but not over generations, setting $p_m(i)$ for bit number $i \in \{1, \ldots, \ell\}$ ($i = 1$ indicates the least significant bit) to a value

$$p_m(i) = \frac{0.3528}{2^{i-1}} \tag{E1.2.6}$$

(iv) a combination of both according to

$$p_m(i, t) = \frac{28}{1905 \times 2^{i-1}} + \frac{0.4026}{2^{t+i-1}}. \tag{E1.2.7}$$

The graphs of these schedules (ii)–(iv) are shown in figures E1.2.1 and E1.2.2 to give an impression of their general form. It is worth noting that the schedule according to (ii) decreases quickly within less than 10 generations to the baseline value.

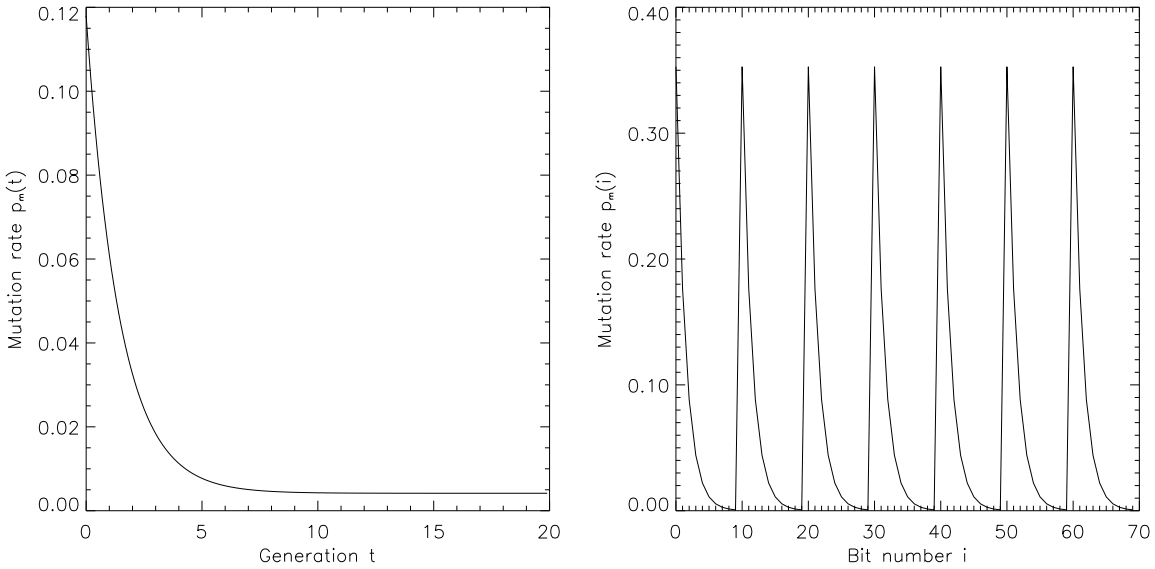


Figure E1.2.1. Mutation rate schedule varying over generation number according to the description given in Fogarty's schedule (ii) (left) and over bit representation according to the description given in his schedule (iii) (right).

For a specific application problem, Fogarty arrived at the conclusion that varying mutation rates over generations and/or across integer representation significantly improves the on-line performance of a genetic algorithm, if evolution is started with a population of all zero bits. Although this result was obtained for a specific experimental setup, Fogarty's investigations serve as an important starting point for other studies about varying mutation rates.

Hesser and Männer (1991, 1992) succeeded in deriving a general expression for a time-varying mutation rate of the form

$$p_m(t) = \left(\frac{c_1}{c_2}\right)^{1/2} \frac{\exp(-c_3 t/2)}{\mu(\ell)^{1/2}} \tag{E1.2.8}$$

which favors an exponentially decreasing mutation rate and seems to confirm Fogarty's findings. Furthermore, equation (E1.2.8) also contains the population size μ as well as the string length ℓ as additional parameters which are relevant for the optimal mutation rate, and the dependence on these parameters shows some correspondence with the empirical finding $p_m \approx 1.75/(\mu(\ell)^{1/2})$ obtained by Schaffer *et al* (1989) by curve fitting of their experimental data. Unfortunately, the constants c_i are generally unknown and can be estimated only for simple cases from heuristic arguments, such that equation (E1.2.8) does not offer a generally useful rule for setting p_m.

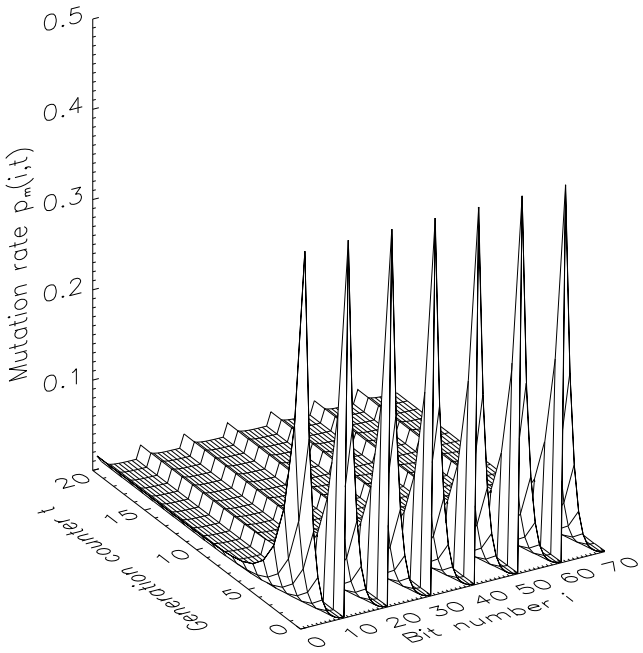


Figure E1.2.2. Mutation rate schedule varying over both bit representation and generation number according to the description given in Fogarty's schedule (iv).

Recently, some results concerning *optimal schedules* of the mutation rate in the cases of simple objective functions and simplified genetic algorithms were presented by Mühlenbein (1992), Bäck (1992a, 1993) and Yanagiya (1993). This work is based on the idea of finding a schedule that maximizes the *convergence velocity* or minimizes the *absorption time* of the algorithm (i.e. the number of iterations until the optimum is found). To facilitate the theoretical analysis, these authors work with the concept of a $(1+\lambda)$- or $(1,\lambda)$-genetic algorithm, (most often, a $(1+1)$-algorithm is considered). Such an algorithm is characterized by a single parent individual, generating λ offspring individuals by mutation. In case of *plus-selection*, the best of parent and offspring is selected for the next generation, while the best of offspring only is selected in case of *comma-selection*. For the simple 'counting ones' objective function $f(b_1 \ldots b_\ell) = \sum_{i=1}^{\ell}$, Bäck (1992a, 1993) demonstrated that a mutation rate schedule starting with $p_m(f(\boldsymbol{b}) = \ell/2) = 1/2$ and decreasing exponentially towards $1/\ell$ as $f(\boldsymbol{b})$ approaches ℓ is optimal, and he presented an approximation for a $(1+1)$-genetic algorithm where

B2.4.2
B2.4, B2.2
C2.4
C2.4

$$p_m(f(\boldsymbol{b})) \approx \frac{1}{2(f(\boldsymbol{b})+1)-\ell} \tag{E1.2.9}$$

defines the mutation rate as a function of $f(\boldsymbol{b})$ at generation t. As a more useful result, however, both Mühlenbein (1992) and Bäck (1993) concluded that a constant mutation rate

$$p_m = 1/\ell \tag{E1.2.10}$$

is almost optimal for a $(1+1)$-genetic algorithm applied to this problem and can serve as a reasonable heuristic rule for any kind of objective function, because it is impossible to derive analytical results for complex functions. This result, however, was already provided by Bremermann *et al* (1966), who used the same approximation method that Mühlenbein used 26 years later. Yanagiya (1993) and Bäck

(1993) also presented optimal mutation rate schedules for more complicated objective functions such as quasi-Hamming-distance functions (the objective function value depends strongly on the Hamming distance from the global optimum), a knapsack problem, and decoding functions mapping binary strings to integers. The resulting optimal mutation rate schedules are often quite irregular and utilize surprisingly large mutation rates, but it seems impossible to draw a general conclusion from these results. Certainly, the above-mentioned rule, $p_m = 1/\ell$, is always a good starting point because it will not perform worse than any smaller mutation rate setting. As the number of offspring individuals increases, however, the optimal mutation rate as well as the associated convergence velocity increase also, but currently no useful analytical results are known for the dependence of the optimal mutation rate on offspring population size (see Bäck 1996, chapter 6).

In addition to deterministic schedules, nondeterministic schedules for controlling the 'amount' (in the sense of continuous step-sizes as well as the probability to invert bits) of mutation are also known in the literature and applied in the context of evolutionary algorithms. Michalewicz (1994) introduced a step-size control mechanism for real-valued vectors, which decreases the amount Δx_i of the modification of an object variable x_i over the number of generations according to

$$\Delta x_i(t, y) = y \left(1 - u^{(1-t/T)^b}\right) \tag{E1.2.11}$$

where $u \sim U([0, 1])$ is a uniform random number, T is the maximum generation number, y is the maximum value of the modification Δx_i, and b is a system parameter determining the degree of dependency on t. Michalewicz (1994, p 101) proposes a value of $b = 5$. In contrast to evolution strategies or evolutionary programming, only a single, randomly chosen (according to a uniform distribution $U(\{1, \ldots, n\})$) object variable is modified when mutation is applied; that is, $m(x_1, \ldots, x_n) = (x_1, \ldots, x_{i-1}, x_i', x_{i+1}, \ldots, x_n)$.

Using equation (E1.2.11), the modification of the selected object variable x_i is performed according to

$$x_i' = \begin{cases} x_i + \Delta x_i(t, \overline{x}_i - x_i) & \text{if } u' = 1 \\ x_i - \Delta x_i(t, x_i - \underline{x}_i) & \text{if } u' = 0 \end{cases} \tag{E1.2.12}$$

where $u' \sim U(\{0, 1\})$ is a uniform random digit, and $\overline{x}_i$, $\underline{x}_i$ denote upper and lower domain bounds of x_i. The plots in figure E1.2.3 show the normalized modification $\Delta x_i(t, y)/y$ as a function of the random variable u for $b = 5$ at two different time stages of a run. Clearly, the possible modification of x_i decreases (quickly, for this value of b) as the generation counter increases.

To facilitate a comparison with a *binary representation* of object variables, Michalewicz (1994) also C1.2

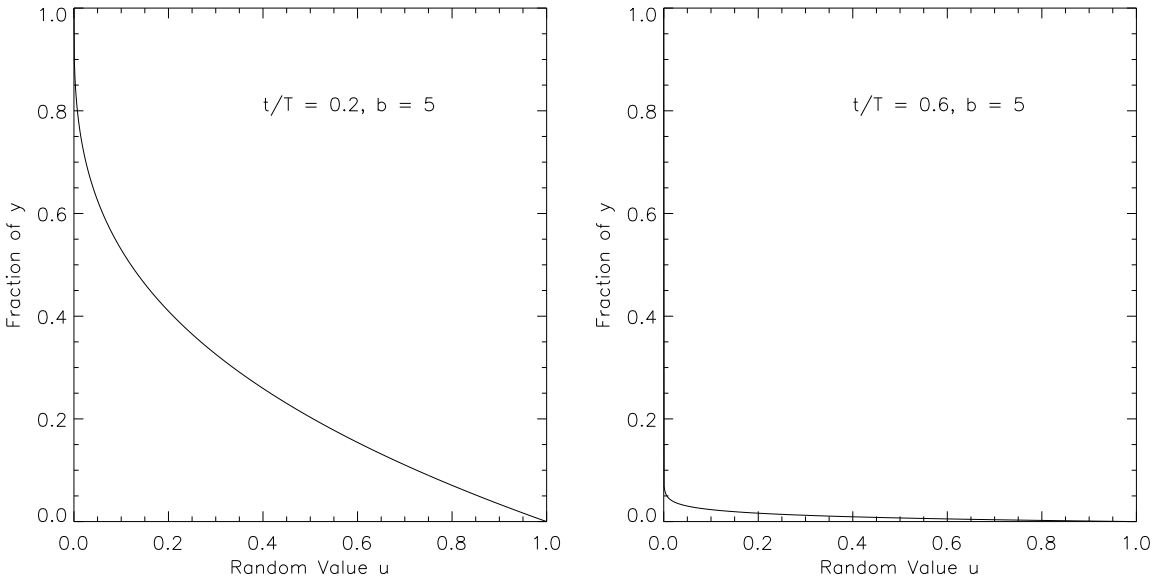


Figure E1.2.3. The plots show the behavior of the normalized modification $\Delta x_i(t, y)/y$ as a function of the random number u according to operator (E1.2.11). In both cases, the default value $b = 5$ was chosen. The left plot ($t/T = 0.2$) demonstrates that the modifications occurring in the early stages of a run are quite large, while later on only small modifications are possible, as shown in the right plot ($t/T = 0.6$).

modeled this mutation operator for the binary space $\{0,1\}^\ell$. Again, the object variable x_i, which should be modified by mutation, is randomly chosen from the n object variables. For the binary representation $(b_1, \ldots, b_{l_x})$ of x_i (with a length of $l_x = 30$ bits per object variable), the mutation operator inverts the value of the bit $b_{\Delta'(t,l_x)}$, where the bit position $\Delta'(t, l_x)$ is defined by

$$\Delta'(t, l_x) = 1 + \begin{cases} \lceil \Delta(t, l_x) \rceil & \text{if } u' = 1 \\ \lfloor \Delta(t, l_x) \rfloor & \text{if } u' = 0 \end{cases} \tag{E1.2.13}$$

and the parameter $b = 1.5$ is chosen to achieve similar behavior as in (E1.2.11) (here, we have to add a value of one because, in contrast to Michalewicz, we consider the least significant bit being indexed by one rather than zero). The effect of this operator is to concentrate mutation on the less significant bits as the generation counter t increases, therefore causing smaller and smaller modifications on the level of decoded object variables. Consequently, this probabilistic mutation rate schedule has some similarity to Fogarty's settings (iii) and (iv), varying the mutation rate both over bit representation and over time.

A comparison of both operators clearly demonstrated a better performance for the operator (E1.2.11) designed for real-valued object variables (Michalewicz 1994, p 102).

E1.2.4 Summary

Optimal setting of the mutation rate or mutation step size(s) in evolutionary algorithms is not an easy task. Use of the self-adaptation technique simplifies the problem by switching to meta-parameters which determine the speed of step size adaptation rather than the step sizes themselves. The new meta-parameters are generally robust and their default settings work well in many cases, though they do not guarantee the fastest adaptation in the general case.

Direct control mechanisms of mutation rate or step size are typically applied in genetic algorithms and their derivates. Usually, a constant mutation rate is utilized, although it is well known that no generally valid best mutation rate exists. It is known, however, that $p_\mathrm{m} = 1/\ell$ is to be preferred over smaller values and can serve as a general guideline if nothing is known about the objective function.

Very little is known concerning mutation rate schedules varying over time or bit representations—except the theoretical results indicating the superiority of schedules depending on the distance to the optimum. In case of the 'counting ones' problem, the decrease of p_m over time is exponential, which seems to be a promising choice also for more difficult objective functions. The alternative way of self-adapting the mutation rate in genetic algorithms, which certainly opens an interesting and promising path towards solving this parameter setting problem, has not yet been exploited in depth (except in some preliminary work by Bäck (1992b)).

References

Bäck T 1992a The interaction of mutation rate, selection, and self-adaptation within a genetic algorithm *Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature (Brussels, 1992)* ed R Männer and B Manderick (Amsterdam: Elsevier) pp 85–94

——1992b Self-adaptation in genetic algorithms *Proc. 1st Eur. Conf. on Artificial Life* ed F J Varela and P Bourgine (Cambridge, MA: MIT Press) pp 263–71

——1993 Optimal mutation rates in genetic search *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 2–8

——1996 *Evolutionary Algorithms in Theory and Practice* (New York: Oxford University Press)

Bäck T and Schwefel H-P 1993 An overview of evolutionary algorithms for parameter optimization *Evolut. Comput.* **1** 1–23

Bremermann H J and Rogson M and Salaff S 1966 Global properties of evolution processes *Natural Automata and Useful Simulations* ed H H Pattec *et al* (Washington, DC: Spartan Books) pp 3–41

De Jong K A 1975 *An analysis of the behavior of a class of genetic adaptive systems* PhD Thesis, University of Michigan

Ebeling W and Engel A and Feistel R 1990 *Physik der Evolutionsprozesse* (Berlin: Akademie-Verlag)

Fogarty T C 1989 Varying the probability of mutation in the genetic algorithm *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, June 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 104–9

Fogel D B 1992 *Evolving Artificial Intelligence* PhD Thesis, University of California, San Diego

Futuyma D J 1990 *Evolutionsbiologie* (Basel: Birkhäuser)

Goldberg D E 1989 *Genetic algorithms in search, optimization and machine learning* (Reading, MA: Addison-Wesley)

Grefenstette J J 1986 Optimization of control parameters for genetic algorithms *IEEE Transactions on Systems, Man and Cybernetics* **SMC–16** 122–8

Hesser J and Männer R 1991 Towards an optimal mutation probability in genetic algorithms *Proc. 1st Workshop on Parallel Problem Solving from Nature (Dortmund, 1990)* (*Lecture Notes in Computer Science 496)* ed H-P Schwefel and R Männer (Berlin: Springer) pp 23–32

——1992 Investigation of the m-heuristic for optimal mutation probabilities *Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature (Brussels, 1992)* ed R Männer and B Manderick (Amsterdam: Elsevier) pp 115–24

Holland J H 1975 *Adaptation in Natural and Artificial Systems* (Ann Arbor, MI: University of Michigan Press)

Michalewicz Z 1994 *Genetic Algorithms + Data Structures = Evolution Programs* (Berlin: Springer)

Mühlenbein H 1992 How genetic algorithms really work: I. Mutation and hillclimbing *Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature (Brussels, 1992)* ed R Männer and B Manderick (Amsterdam: Elsevier) pp 15–25

Rechenberg I 1994 *Evolutionsstrategie '94* Werkstatt Bionik und Evolutionstechnik, vol 1 (Stuttgart: Frommann–Holzboog)

Saravanan N and Fogel D B 1994a Learning of strategy parameters in evolutionary programming: an empirical study *Proc. 3rd Ann. Conf. on Evolutionary Programming (San Diego, CA, February 1994)* ed A V Sebald and L J Fogel (Singapore: World Scientific) pp 269–80

——1994b Evolving neurocontrollers using evolutionary programming *Proc. 1st IEEE Conf. on Evolutionary Computation (Orlando, FL, June 1994)* (Piscataway, NJ: IEEE) pp 217–22

Schaffer J D, Caruana R A, Eshelman L J and Das R 1989 A study of control parameters affecting online performance of genetic algorithms for function optimization *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, June 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 51–60

Schwefel H-P 1977 *Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie* (*Interdisciplinary Systems Research 26*) (Basel: Birkhäuser)

Schwefel H-P and Rudolph G 1995 Contemporary evolution strategies *Advances in Artificial Life :(Proc. 3rd Int. Conf. on Artificial Life)* (*Lecture Notes in Artificial Intelligence 929*) ed F Morán *et al* (Berlin: Springer) pp 893–907

Yanagiya M 1993 A simple mutation-dependent genetic algorithm *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) p 695

E1.3 Recombination parameters

William M Spears

Abstract

One operator that is often used in evolution strategies, genetic algorithms, and genetic programming is recombination, where material from two (or more) parents is used to create new offspring. There are numerous ways to implement recombination. This section will focus mainly on recombination operators that construct potentially useful solutions to a problem from smaller components (often called *building blocks*). This section gives an overview of some of the motivation, issues, theory, and heuristics for building block recombination.

E1.3.1 General background

Although Holland (1975) was not the first to suggest *recombination* in an evolutionary algorithm (EA) (see e.g. Fraser 1957, Fogel *et al* 1966), he was the first to place theoretical emphasis on this operator. This emphasis stemmed from his work in adaptive systems, which resulted in the field of *genetic algorithms* (GAs) and *genetic programming*. According to Holland, an adaptive system must persistently test and incorporate structural properties associated with better performance. The object, of course, is to find *new* structures which have a high probability of improving performance significantly.

C3.3 B1.2 B1.5.1

Holland concentrated on *schemata*, which provide a basis for associating combinations of attributes with potential for improving current performance. To see this, let us consider the schema AC##, defined over a fixed length chromosome of four genes, where each gene can take on one of three alleles {A, B, C}. If # is defined to be a 'don't care' (i.e. wildcard) symbol, the schema AC## represents all chromosomes that have an A for their first allele and a C for their second. Since each of the # symbols can be filled in with any one of the three alleles, this schema represents $3^2 = 9$ chromosomes.

B2.5

Suppose every chromosome has a well-defined fitness value (also called *utility* or *payoff*). Now suppose there is a population of P individuals, p of which are members of the above schema. The *observed average fitness* of that schema is the average fitness of these p individuals in that schema. It is important to note that these individuals will also be members of other schemata, thus the population of P individuals contains instances of a large number of schemata (all of which have some observed fitness). Holland (1975) stated that a good heuristic is to generate new instances of these schemata whose observed fitness is higher than the average fitness of the whole population, since instances of these schemata are likely to exhibit superior performance.

Suppose the schema AC## does in fact have a high observed fitness. The heuristic states that new samples (instances) of that schema should be generated. Selection (reproduction) does not produce new samples—but recombination can. The key aspect of recombination is that if one recombines two individuals that start with AC, their offspring must also start with AC. Thus one can retain what appears to be a promising building block (AC##), yet continue to test that building block in new contexts.

As stated earlier, recombination can be implemented in many different ways. Some forms of recombination are more appropriate for certain problems than are others. According to Booker (1992) it is thus useful to characterize the biases in recombination operators, recognize when these biases are correct or incorrect (for a given problem or problem class), and recover from incorrect biases when possible. Sections E1.3.2 and E1.3.3 summarize much of the work that has gone into characterizing the biases of

various recombination operators. Historically, most of the earlier recombination operators were designed to work on low-level 'universal' representations, such as the fixed-length low-cardinality representation shown above. In fact, most early GAs used just simple bitstring representations. A whole suite of recombination operators evolved from that level of representation. Section E1.3.2 focuses on such 'bit-level' recombination operators. Recent work has focused more on problem-class-specific representations, with recombination operators designed primarily for these representations. Section E1.3.3 focuses on the more problem-class-specific recombination operators.

Section E1.3.4 summarizes some of the mechanisms for recognizing when biases are correct or incorrect, and recovering from incorrect biases when possible. The conclusion outlines some design principles that are useful when creating new recombination operators. Ideally, one would like firm and precise practical rules for choosing what form and rate of recombination to use on a particular problem; however, such rules have been difficult to formulate. Thus this section concentrates more on heuristics and design principles that have often proved useful.

E1.3.2 Genotypic-level recombination

E1.3.2.1 Theory

Holland (1975) provided one of the earliest analyses of a *recombination operator*, called one-point recombination. Suppose there are two parents: ABCD and AABC. Randomly select one point at which to separate (*cut*) both parents. For example, suppose they are cut in the middle (AB|CD and AA|BC). The offspring are created by swapping the tail (or head) portions to yield ABBC and AACD. Holland analyzed one-point recombination by examining the probability that various schemata will be disrupted when undergoing recombination. For example, consider the two schemata AA## and A##A. Each schema can be disrupted only if the *cut point* falls between its two As. However, this is much more likely to occur with the latter schema (A##A) than the former (AA##). In fact, the probability of disrupting either schema is proportional to the distance between the As. Thus, one-point recombination has the bias that it is much more likely to disrupt 'long' schemata than 'short' schemata, where the length of a schema is the distance between the first and the last *defining position* (a nonwildcard). C3.3.1

De Jong (1975) extended this analysis to include so-called *n-point* recombination. In n-point recombination n cut points are randomly selected and the genetic material between cut points is swapped. For example, with two-point recombination, suppose the two parents ABCD and AABC are cut as follows: A|BC|D and A|AB|C. Then the two offspring are AABD and ABCC. De Jong noted that two-point (or n-point where n is even) recombination is less likely to disrupt 'long' schemata than one-point (or n-point where n is odd) recombination.

Syswerda (1989) introduced a new form of recombination called *uniform* recombination. Uniform recombination does not use *cut points* but instead creates offspring by deciding, for each allele of one parent, whether to swap that allele with the corresponding allele in the other parent. That decision is made using a coin flip (i.e. the swap is made 50% of the time). Syswerda compared the probability of schema disruption for one-point, two-point, and uniform recombination. Interestingly, while uniform recombination is somewhat more disruptive of schemata than one-point and two-point, it does not have a length bias (i.e. the length of a schema does not affect the probability of disruption). Also, Syswerda showed that the more disruptive nature of uniform recombination can be viewed in another way—it is more likely to construct instances of new schemata than one-point and two-point recombination.

De Jong and Spears (1992) verified Syswerda's results and introduced a parameterized version of uniform recombination (where the probability of swapping could be other than 50%). Lowering the swap probability of uniform recombination allows one to lower disruption as much as desired, while maintaining the lack of length bias. Finally, De Jong and Spears characterized recombination in terms of two other measures: *productivity* and *exploration power*. The productivity of a recombination operator is the probability that it will generate offspring that are different from their parents. More disruptive recombination operators are more productive (and vice versa). An operator is more *explorative* if it can reach a larger number of points in the space with one application of the operator. Uniform recombination is the most explorative of the recombination operators since, if the Hamming distance between two parents is h (i.e. h loci have different alleles), uniform recombination can reach any of 2^h points in one application of the operator. Moon and Bui (1994) independently performed a similar analysis. Although mathematically equivalent, this analysis emphasized 'clusters' of defining positions within schemata, as opposed to lengths.

Eshelman *et al* (1989) considered other characterizations of recombination bias. They introduced two biases, the *positional* and *distributional bias*. A recombination operator has positional bias to the extent that the creation of any new schema by recombining existing schemata is dependent upon the location of the alleles in the chromosome. This is similar to the length bias introduced above. A recombination operator has distributional bias to the extent that the amount of material that is expected to be exchanged is distributed around some value or values ranging from 1 to $L-1$ alleles (where the chromosome is composed of L genes), as opposed to being uniformly distributed. For example, one-point recombination has high positional and no distributional bias, while two-point recombination has slightly lower positional bias and still no distributional bias. Uniform recombination has no positional bias but high distributional bias because the amount of material exchanged is binomially distributed. Later, Eshelman and Schaffer (1994) refined their earlier study and introduced *recombinative* bias, which is related to their older distributional bias. They also introduced *schema* bias, which is a generalization of their older positional bias.

Booker (1992) tied the earlier work together by characterizing recombination operators via their recombination distributions, which describe the probability of all possible recombination events. The recombination distributions were used to rederive the disruption analysis of De Jong and Spears (1992) for n-point and parameterized uniform recombination, as well as to calculate precise values for the distributional and positional biases of recombination. This reformulation allowed Booker to detect a symmetry in the positional bias of n-point recombination around $n = L/2$, which corrected a prediction made by Eshelman *et al* (1989) that positional bias would continue to increase as n increases.

E1.3.2.2 Heuristics

The sampling arguments and the characterization of biases that has just been presented have motivated a number of heuristics for how to use recombination and how to choose which recombination to use.

Booker (1982) considered implementations of recombination from the perspective of trying to improve overall performance. The motivation was that allele loss from the population could hurt the sampling of coadapted sets of alleles (schemata). In the earliest implementations of GAs one offspring of a recombination event would be thrown away. This was a source of allele loss, since, instead of transmitting all alleles from both parents to the next generation, only a subset was transmitted. The hypothesis was that allele loss rates would be greatly decreased by saving both offspring. That hypothesis was confirmed empirically. There was also some improvement in on-line (average fitness of all samples) and off-line (average fitness of the best samples) performance on the De Jong (1975) test suite, although the off-line improvement was negligible.

Booker (1987) also pointed out that, due to allele loss, recombination is less likely to produce children different from their parents as a population evolves. This effectively reduces the sampling of new schemata. To counteract this Booker suggested a more explorative version of recombination, termed *reduced surrogate* recombination, that concentrates on those portions of a chromosome in which the alleles of two parents are not the same. This ensures that a new sample is created. For example, suppose two parents are ABCD and ADBC. Then if one uses one-point recombination, and the cut point occurs immediately after the A, the two offspring would be identical to the parents. Reduced surrogate recombination would ensure that the cut point was further to the right.

It has been much more difficult to come up with heuristics for choosing which recombination operator to use in a given situation. Syswerda (1989), however, noted that one nice aspect of uniform recombination is that, due to its lack of length bias, it is not affected by the presence of irrelevant alleles in a representation. Nor is it affected by the position of the relevant alleles on the chromosome. Thus, for those problems where little information is available concerning the relevance of alleles or the length of building blocks, uniform recombination is a useful default.

De Jong and Spears (1990) tempered this view somewhat, by including interactions with *population size*. Their heuristic was that disruption is most useful when the population size is small or when the population is almost homogeneous. They argued that more disruptive recombination operators (such as 0.5 uniform recombination, or n-point recombination where $n > 2$) should be used when the population size is small relative to the problem size, and less disruptive recombinations operators (such as two-point, or uniform recombination with a swap probability less than 50%) should be used when the population size is large relative to the problem size. De Jong and Spears demonstrated this with a series of experiments in which the population size was varied. E1.1

Schaffer *et al* (1989) made a similar observation. They concentrated on the *recombination rate*, which is the percentage of the population to undergo recombination. They observed that high recombination rates

are best with small populations, a broad range of recombination rates are tolerated at medium population sizes, and only low recombination rates are suggested for large population sizes.

Finally, Eshelman and Schaffer (1994) have attempted to match the biases of recombination operators with various problem classes and GA behavior. They concluded with two heuristics. The first was that high schema bias can lead to hitchhiking, where the EA exploits spurious correlations between schemata that contribute to performance and other schemata that do not. They recommended using a high-recombinative-bias and low-schema-bias recombination to combat premature convergence (i.e. loss of genetic diversity) due to hitchhiking. The second heuristic was that high recombinative bias can be detrimental in trap problems.

E1.3.3 Phenotypic-level recombination

E1.3.3.1 Theory

Thus far the focus has been on fixed-length representations in which each gene can take on one of a discrete set of alleles (values). Schemata were then defined, each of which represent a set of chromosomes (the chromosomes that match alleles on the defining positions of the schema). However, there are problems that do not match these representations well. In these cases new representations, recombination operators, and theories must be developed.

For example, a common task is the optimization of some real-valued function of real values. Of course, it is possible to code these real values as bitstrings in which the degree of granularity is set by choosing the appropriate number of bits. At this point conventional schema theory may be applied. However, there are difficulties that arise using this representation. One is the presence of *Hamming cliffs*, in which large changes in the binary encoding are required to make small changes to the real values. The use of Gray codes does not totally remove this difficulty. Standard recombination operators also can have the effect of producing offspring far removed (in the real-valued sense) from their parents (Schwefel 1995). An alternative representation is to simply use chromosomes that are *real-valued vectors*. In this C1.3 case a more natural recombination operator averages (blends) values within two parent vectors to create an offspring vector. This has the nice property of creating offspring that are near the parents. (See the work of Davis (1991), Wright (1991), Eshelman, and Schaffer (1992), Schwefel (1995), Peck and Dhawan (1995), Beyer (1995), and Arabas *et al* (1995) for other recombination operators that are useful for real-valued vectors. One recombination operator of note is discrete recombination, which is the analog of uniform recombination on real-valued variables.)

Recently, three theoretical studies analyzed the effect of recombination using real-valued representations. Peck and Dhawan (1995) showed how various properties of recombination operators can influence the ability of the EA to converge to the global optima. Beyer (1995) concluded that an important role of recombination in this context is genetic repair, diminishing the influence of harmful mutations. Eshelman and Schaffer (1992) analyzed this particular representation by restricting the parameters to be integer ranges. Their *interval schemata* represent subranges. For example, if a parameter has range [0, 2], it has the following interval schemata: [0], [1], [2], [0, 1], [1, 2], and [0, 2]. The chromosomes (1 1 2) and (2 1 2) are instances of the interval schema ([1, 2] [1] [2]). Long interval schemata are more general and correspond roughly to traditional schemata that contain a large number of #s. Eshelman and Schaffer used *interval schemata* to help them predict the failure modes of various real-valued recombination operators.

Another class of important tasks involves *permutation* or ordering problems, in which the ordering of C3.3.3 alleles on the chromosome is of primary importance. A large number of recombination operators have been suggested for these tasks, including partially mapped recombination (Goldberg and Lingle 1985), order recombination (Davis 1985), cycle recombination (Oliver *et al* 1987), edge recombination (Starkweather *et al* 1991), and the group recombination of Falkenauer (1994). Which operators work best depend on the objective function.

A classic permutation problem is the traveling salesman problem (TSP). Consider a TSP of four cities {A, B, C, D}. It is important to note that there are only 4! possible chromosomes, as opposed to 4^4 (e.g. the chromosome AABC is not valid). Also, note that schema ##BC does not have the same meaning as before, since the alleles that can be used to fill in the #s now depend on the alleles in the defining positions (e.g. a B or a C can not be used in this case). This led Goldberg and Lingle (1985) to define o-schemata, in which the 'don't cares' are denoted with !s. An example of an o-schema is !!BC, which defines the subset of all orderings that have BC in the third and fourth positions. For this example there

are only two possible orderings, ADBC and DABC. Goldberg considered these to be *absolute* o-schemata, since the absolute position of the alleles is of importance. An alternative would be to stress the relative positions of the alleles. In this case what is important about !!BC is that B and C are adjacent—!!BC, !BC!, and BC!! are now equivalent o-schemata. One nice consequence of the invention of o-schemata is that a theory similar to that of the more standard schema theory can be developed. The interested reader is encouraged to see the article by Oliver *et al* (1987) for a nice example of this, in which various recombination operators are compared via an o-schema analysis.

There has also been some work on recombination for *finite-state machines* (Fogel *et al* 1966), variable-length chromosomes (Smith 1980, De Jong *et al* 1993), chromosomes that are *LISP expressions* (Fujiki and Dickinson 1987, Koza 1994, Rosca 1995), chromosomes that represent strategies (i.e. rule sets) (Grefenstette *et al* 1990), and recombination for multidimensional chromosomes (Kahng and Moon 1995). Some theory has recently been developed in these areas. For example, Bui and Moon (1995) developed some theory on multidimensional recombination. Also, Radcliffe (1991) generalized the notion of schemata to sets he refers to as *formae*.

C1.5 C3.3.5 B2.5.5

E1.3.3.2 Heuristics

Due to the prevalence of the traditional bitstring representation in GAs, less work has concentrated on recombination operators for higher-level representations, and there are far fewer heuristics. The most important heuristic is that recombination must identify and combine meaningful building blocks of chromosomal material. Put another way, 'recombination must take into account the interaction among the genes when generating new instances' (Eshelman and Schaffer 1992). The conclusion of this section provides some guidance in how to achieve this.

E1.3.4 Control of recombination parameters

As can be seen from the earlier discussion, there is very little theory or guidance on how to choose *a priori* which recombination operator to use on a new problem. There is also very little guidance on how to choose how often to apply recombination (often referred to as the *recombination rate*). There have been three approaches to this problem, referred to as *static*, *predictive*, and *adaptive* approaches.

E1.3.4.1 Static techniques

The simplest approach is to assume that one particular recombination operator should be applied at some static rate for all problems. The static rate is estimated from a set of empirical studies, over a wide variety of problems, population sizes, and mutation rates. Three studies are of note. De Jong (1975) studied the on-line and off-line performance of a GA on the De Jong test suite and recommended a recombination rate of 60% for one-point recombination (i.e. 60% of the population should undergo recombination). Grefenstette (1986) studied on-line performance of a GA on the De Jong test suite and recommended that one-point recombination be used at the higher rate of 95%. In the most recent study, Eshelman *et al* (1989) studied the mean number of evaluations required to find the global optimum on the De Jong test suite and recommended an intermediate rate of 70% for one-point recombination. Each of these settings for the recombination rate is associated with particular settings for mutation rates and population sizes, so the interested reader is encouraged to consult these references for more complete information.

E1.3.4.2 Predictive techniques

In the static approach it is assumed that some fixed recombination rate (and recombination operator) is reasonable for a large number of problems. However, this will not be true in general. The predictive approaches are designed to predict the performance of recombination operators (i.e. to recognize when the recombination bias is correct or incorrect for the problem at hand).

Manderick *et al* (1991) computed fitness correlation coefficients for different recombination operators on various problems. Since they noticed a high correlation between operators with high correlation coefficients and good GA performance their approach was to choose the recombination operator with the highest correlation coefficient. The approach of Grefenstette (1995) was similar in spirit to that of Manderick *et al*. Grefenstette used a *virtual* GA to compute the past performance of an operator as an estimate of the future performance of an operator. By running the virtual GA with different recombination operators, Grefenstette estimated the performance of those operators in a real GA.

Altenberg (1994) and Radcliffe (1994) have proposed different predictive measures. Altenberg proposed using an alternative statistic referred to as the *transmission function in the fitness domain*. Radcliffe proposed using the *fitness variance of formae* (generalized schemata). Thus far all approaches have shown considerable promise.

E1.3.4.3 Adaptive techniques

In both the static and predictive approaches the decision as to which recombination operator and the rate at which it should be applied is made prior to actually running the EA. However, since these approaches can make errors (i.e. choose nonoptimal recombination operators or rates), a natural solution is to make these choices adaptive. Adaptive approaches are designed to recognize when bias is correct or incorrect, and recover from incorrect biases when possible. For the sake of exposition adaptive approaches will be divided into *tag-based* and *rule-based*. As a general rule, tag-based approaches attach extra information to a chromosome, which is both evolved by the EA and used to control recombination. The rule-based approaches generally adapt recombination using control mechanisms and data structures that are external to the EA.

One of the earliest tag-based approaches was that of Rosenberg (1967). In this approach integers x_i ranging from zero to seven were attached to each locus. The recombination site was chosen from the probability distribution defined over these integers, $p_i = x_i / \sum x_i$, where p_i represented the probability of a cross at site i.

Schaffer and Morishima (1987) used a similar approach, by adjusting the points at which recombination was allowed to cut and splice material. They accomplished this by appending an additional L bits to L-bit individuals. These appended bits were used to determine *cut points* for each locus (a 'one' denoted a cut point while a 'zero' indicated the lack of a cut point). If two individuals had n distinct cut points, this was analogous to using a particular instantiation of n-point recombination.

Levenick (1995) also had a similar approach. Recombination was implemented by replicating two parents from one end to the other by iterating the following algorithm:

(i) Copy one bit from parent1 to child1
(ii) Copy one bit from parent2 to child2
(iii) With some base probability P_b perform a recombination: swap the roles of the children (so subsequent bits come from the other parent).

Levenick inserted a metabit before each bit of the individual. If the metabit was 'one' in both parents recombination occurred with probability P_b, else recombination occurred with a reduced probability P_r. The effect was that the probability of recombination could be reduced from a maximum of P_b to a minimum of P_r. Levenick claimed that this method improved performance in those cases where the population did not converge too rapidly.

Arabas *et al* (1995) experimented with adaptive recombination in an evolution strategy. Each chromosome consisted of L real-valued parameters combined with an additional L control parameters. In a standard evolution strategy these extra control parameters are used to adapt mutation. In this particular study the control parameters were also used to adapt recombination, by concentrating offspring around particular parents. Empirical results on four classes of functions were encouraging.

Angeline (1996) evolved *LISP expressions*, and associated a recombination probability with each node in the LISP expressions. These probabilities evolved and controlled the application of recombination. Angeline investigated two different adaptive mechanisms based on this approach and reported that the adaptive mechanisms outperformed standard recombination on three test problems. C3.3.5

Spears (1995) used a simple approach in which one extra *tag bit* was appended to every individual. C6.2.5 The tag bits were used to control the use of two-point and uniform recombination in the following manner:

```
if (parent1[L + 1] = parent2[L + 1] = 1)
then two-point-recombination(parent1, parent2)
else if (parent1[L + 1] = parent2[L + 1] = 0)
then uniform-recombination(parent1, parent2)
else if (rand(0, 1) < 0.5)
    then two-point-recombination(parent1, parent2)
    else uniform-recombination(parent1, parent2)
```

Spears compared this adaptive approach on a number of different problems and population sizes, and found that the adaptive approach always had a performance intermediate between the best and worst of the two single recombination operators.

Rule-based approaches use auxiliary data structures and statistics to control recombination. The simplest of these approaches use hand-coded rules that associate various statistics with changes in the recombination rate. For example, Wilson (1986) examined the application of GAs to *classifier systems* B1.5.2 and defined an entropy measure over the population. If the change in entropy was sufficiently positive or negative, the probability of recombination was decreased or increased respectively. The idea was to introduce more variation by increasing the recombination rate whenever the previous variation had been 'absorbed'. Booker (1987) considered the performance of GAs in function optimization and measured the percentage of the current population that produced offspring. Every percentage change in that measure was countered with an equal and opposite percentage change in the recombination rate.

Srinivas and Patnaik (1994) considered measures of fitness performance and used those measures to estimate the distribution of the population. They increased the probability of recombination (P_c) and the probability of mutation (P_m) when the population was stuck at local optima and decreased the probabilities when the population was scattered in the solution space. They also considered the need to preserve 'good' solutions of the population and attempted this by having lower values of P_c and P_m for high-fitness solutions and higher values of P_c and P_m for low-fitness solutions.

Hong *et al* (1995) had a number of different schemes for adapting the use of multiple recombination operators. The first scheme was defined by the following rule: if both parents were generated via the same recombination operator, apply that recombination operator, else randomly select (with a coin flip) which operator to use. This first scheme was very similar to that of Spears (1995) but does not use tag bits. Their second scheme was the opposite of the first: if both parents were generated via the same recombination operator, apply some other recombination operator, else randomly select (with a coin flip) which operator to use. Their third scheme used a measure called an *occupancy rate*, which was the number of individuals in a population that were generated by a particular recombination (divided by the population size). For k recombination operators, their third scheme tried to balance the occupancy rate of each recombination operator around $1/k$. In their experiments the second and third schemes outperformed the first (although this was not true when they tried uniform and two-point recombination).

Eshelman and Schaffer (1994) provided a switching mechanism to decide between two recombination operators that often perform well, HUX (a variant of uniform crossover where exactly half of the differing bits are swapped at random) and SHX (a version of one-point recombination in which the positional bias has been removed). Their GA uses restarts—when the population is (nearly) converged the converged population is partially or fully randomized and seeded with one copy of the best individual found so far (the *elite* individual). During any convergence period between restarts (including the period leading up to the first restart), either HUX or SHX is used but not both. HUX is always used during the first two convergences. Subsequently, three rules are used for switching recombination operators:

(i) SHX is used for the next convergence if during the prior convergence no individual is found that is as good as the elite individual.

(ii) HUX is used for the next convergence if during the prior convergence no individual is found that is better than the elite individual, but at least one individual is found that is as good as the elite individual.

(iii) No change in the operator is made if during the prior convergence a new best individual is found (which will replace the old elite individual).

These methods had fairly simple rules and data structures. However, more complicated techniques have been attempted. Davis (1989) provided an elaborate bookkeeping method to reward recombination operators that produced good offspring or set the stage for this production. When a new individual was added to the population, a pointer was established to its parent or parents, and a pointer was established to the operator that created the new individual. If the new individual was better than the current best member of the population, the amount it was better was stored as its *local delta*. Local deltas were passed back to parents to produce *inherited deltas*. *Derived deltas* were the sums of local and inherited deltas. Finally, the *operator delta* was the sum of the derived deltas of the individuals it produced, divided by the number of individuals produced. These operator deltas were used to update the probability that the operator would be fired.

White and Oppacher (1994) used finite-state automata to identify groups of bits that should be kept together during recombination (an extension of uniform recombination). The basic idea was to learn from previous recombination operations in order to minimize the probability that highly fit schemata would be disrupted in subsequent recombination operations. The basic bitstring representation was augmented at each bit position with an automaton. Each state of the automaton mapped to a probability of recombination for that bitstring location—roughly, given N states, then the probability p_i associated with state i was i/N. Some of the heuristics used for updating the automaton state were:

(i) if offspring fitness $>$ fitness of the father(mother)
then reward those bits that came from the father(mother)
(ii) if offspring fitness $<$ fitness of the father(mother)
then penalize those bits that came from the father(mother).

There were other rules to handle offspring of equal fitness. A reward implied that the automaton moved from state i to state $i + 1$, and a penalty implied that the automaton moved from state i to state $i - 1$.

Julstrom (1995) used an *operator tree* to fire recombination more often if it produced children of superior fitness. With each individual was an operator tree—a record of the operators that generated the individual and its ancestors. If a new individual had fitness higher than the current population median fitness, the individual's operator tree was scanned to compute the credit due to recombination (and mutation). A queue recorded the credit information for the most recent individuals. This information was used to calculate the probability of recombination (and mutation).

Finally, Lee and Takagi (1993) evolved *fuzzy rules* for GAs. The fuzzy rules had three input variables based on fitness measures: D2

(i) x = average fitness/best fitness
(ii) y = worst fitness/average fitness
(iii) z = change in fitness.

The rules had three possible outputs dealing with population size, recombination rate, and mutation rate. All of the variables could take on three values {small, medium, big}, with the semantics of those values determined by membership functions. The rules were evaluated by running the GA on the De Jong test suite and different rules were obtained for on-line and off-line performance. 51 rules were obtained in the fuzzy rulebase. Of these, 18 were associated with recombination. An example was: if (x is small) and (y is small) and (z is small) then the change in recombination rate is small.

In summary, the fixed-recombination-rate approaches are probably the least successful, but provide reasonable guesses for parameter settings. They also are reasonable settings for the initial stages of the predictive and adaptive approaches. The predictive approaches have had success and appear very promising. The adaptive approaches also have had some success. However, as Spears (1995) indicated, a common difficulty in the evaluation of the adaptive approaches has been the lack of adequate control studies. Thus, although the approaches may show signs of adaptation, it is not clear that adaptation is the cause of performance improvement.

E1.3.5 Discussion

The successful application of recombination (or any other operator) involves a close link with the operator, the representation, and the objective function. This has been outlined by Peck and Dhawan (1995), who

emphasized similarity—one needs to exploit similarities between previous high-performance samples, and these similar samples must have similar objective function values often enough for the algorithm to be effective. Goldberg (1989) and Falkenauer (1994) make a similar point when they refer to *meaningful building blocks*. This has led people to outline various issues that must be considered when designing a representation and appropriate recombination operators.

De Jong (1975) outlined several important issues with respect to representation. First, 'nearbyness' should be preserved, in which small changes in a parameter value should come about from small changes in the representation for that value. Thus, binary encodings of real-valued parameters are problematic, since Hamming cliffs separate parameters that are near in the real-valued space and standard recombination operators can produce offspring far removed (in the real-valued sense) from their parents. Second, it is generally better to have context-insensitive representations, in which the legal values for a parameter do not depend on the values of other parameters. Finally, it is generally better to have context-insensitive interpretations of the parameters, in which the interpretation of some parameter value does not depend on the values of the other parameters. These last two concerns often arise in permutation or ordering problems, in which the values of the leftmost parameters influence both the legal values and the interpretation of these values for the rightmost parameters. For example, the encoding of the TSP problem presented earlier is context sensitive, and standard recombination operators can produce invalid offspring when using the representation. An alternative representation could be one in which the first parameter specifies which of the N cities should be visited first. Having deleted that city from the list of cities, the second parameter always takes on a value in the range $1 \ldots N-1$, specifying by position on the list which of the remaining cities is to be visited second, and so on. For example, suppose there are four cities {A, B, C, D}. The representation of the tour BCAD is (2 2 1 1) because city B is the second city in the list {A, B, C, D}, C is the second city in the list {A, C, D}, A is the first city in the list {C, D} and D is the first city in the list {D}. This representation is context insensitive and recombination of two tours always yields a valid tour. However, it has a context-sensitive interpretation, since gene values to the right of a recombination cut point specify different subtours in the parent and the offspring.

Radcliffe (1991) outlined three design principles for recombination. First, recombination operators should be *respectful*. Respect occurs if crossing two instances of any forma (a generalization of schema) must produce another instance of that forma. For example, if both parents have blue eyes then all their children must have blue eyes. This principle holds for any standard recombination on bitstrings. Second, recombination should *properly assort* formae. This occurs if, given instances of two compatible formae, it must be possible to cross them to produce an offspring which is an instance of both formae. For example, if one parent has blue eyes and the other has brown hair, it must be possible to recombine them to produce a child with blue eyes and brown hair. This principle is similar to what others called exploratory power—e.g. uniform recombination can reach all points in the subspace defined by the differing bits (in one application), while n-point recombination cannot. Thus n-point recombination does not properly assort, while uniform recombination does. Finally, recombination should *strictly transmit*. Strict transmission occurs if every allele in the child comes from one parent or another. For example, if one parent has blue eyes and the other has brown eyes, the child must have blue or brown eyes. All standard recombination operators for bitstrings strictly transmit genes.

B2.5.2.2

All of this indicates that the creation and successful application of recombination operators is not 'cut and dried', nor a trivial pursuit. Considerable effort and thought is required. However, if one uses the guidelines suggested above as a first cut, success is more likely.

Acknowledgements

I thank Diana Gordon, Chad Peck, Mitch Potter, Ken De Jong, Peter Angeline, David Fogel, and the George Mason University GA Group for helpful comments on organizing this paper.

References

Altenberg L 1994 The schema theorem and Price's theorem *Proc. 3rd Foundations of Genetic Algorithms Workshop* ed M Vose and D Whitley (San Mateo, CA: Morgan Kaufmann) pp 23–49

Angeline P 1996 Two self-adaptive crossover operations for genetic programming *Adv. Genet. Programming* **2** 89–110

Arabas J, Mulawka J and Pokrasniewicz J 1995 A new class of the crossover operators for the numerical optimization *Proc. 6th Int. Conf. on Genetic Algorithms (Pittsburgh, PA, 1995)* ed L J Eshelman (San Mateo, CA: Morgan Kaufmann) 42–8

Beyer H-G 1995 Toward a theory of evolution strategies: on the benefits of sex—the $(\mu/\mu, \lambda)$ theory *Evolutionary Computation* **3** 81–112

Booker L B 1982 *Intelligent Behavior as an Adaptation to the Task Environment* PhD Dissertation, University of Michigan

——1987 Improving search in genetic algorithms *Genetic Algorithms and Simulated Annealing* ed L Davis (Los Altos, CA: Morgan Kaufmann) pp 61–73

——1992 Recombination distributions for genetic algorithms *Proc. 2nd Foundations of Genetic Algorithms Workshop* ed D Whitley (San Mateo, CA: Morgan Kaufmann) pp 29–44

Bui T and Moon B 1995 On multi-dimensional encoding/crossover *Proc. 6th Int. Conf. on Genetic Algorithms (Pittsburgh, PA, 1995)* ed L J Eshelman (San Mateo, CA: Morgan Kaufmann) pp 49–56

Davis L 1985 Applying adaptive algorithms in epistatic domains *Proc. Int. Joint Conf. on Artificial Intelligence*

——1989 Adapting operator probabilities in genetic algorithms *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, 1989)* ed J Schaffer (San Mateo, CA: Morgan Kaufmann) pp 61–9

——1991 Hybridization and numerical representation *Handbook of Genetic Algorithms* ed L Davis (New York: Van Nostrand Reinhold) pp 61–71

De Jong K 1975 *Analysis of the Behavior of a Class of Genetic Adaptive Systems* PhD Dissertation, University of Michigan

De Jong K and Spears W 1990 An analysis of the interacting roles of population size and crossover in genetic algorithms *Proc. Int. Conf. on Parallel Problem Solving from Nature* ed H-P Schwefel and R Männer (Berlin: Springer) pp 38–47

——1992 A formal analysis of the role of multi-point crossover in genetic algorithms *Annals of Mathematics and Artificial Intelligence* (Switzerland: Baltzer) **5 1** 1–26

De Jong K, Spears W and Gordon D 1993 Using genetic algorithms for concept learning *Machine Learning* **13** 161–88

Eshelman L and Schaffer D 1992 Real-coded genetic algorithms and interval-schemata *Proc. 2nd Foundations of Genetic Algorithms Workshop* ed D Whitley (San Mateo, CA: Morgan Kaufmann) pp 187–202

——1994 Productive recombination and propagating and preserving schemata *Proc. 3rd Foundations of Genetic Algorithms Workshop* ed M Vose and D Whitley (San Mateo, CA: Morgan Kaufmann) pp 299–313

Eshelman L, Caruana R and Schaffer D 1989 Biases in the crossover landscape *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, 1989)* ed J Schaffer (San Mateo, CA: Morgan Kaufmann) pp 10–19

Falkenauer E 1994 A new representation and operators for genetic algorithms applied to grouping problems *Evolutionary Computation* (Cambridge, MA: MIT Press) **2 2** 123–44

Fogel L, Owens A and Walsh M 1966 *Artificial Intelligence through Simulated Evolution* (New York: Wiley)

Fraser A 1957 Simulation of genetic systems by automatic digital computers I Introduction *Aust. J. Biol. Sci.* **10** 484–91

Fujiki C and Dickinson J 1987 Using the genetic algorithm to generate lisp source code to solve the prisoner's dilemma *Proc. 2nd Int. Conf. on Genetic Algorithms* (Pittsburgh, PA, 1987) ed J J Grefenstette (Hillsdale, NJ: Erlbaum) pp 236–40

Goldberg D 1989 *Genetic Algorithms in Search Optimization and Machine Learning* (Reading, MA: Addison-Wesley)

Goldberg D and Lingle R 1985 Alleles loci and the traveling salesman problem *Proc. 1st Int. Conf. on Genetic Algorithms and their Applications (Pittsburgh, PA, 1985)* ed J J Grefenstette (Hillsdale, NJ: Erlbaum) pp 154–9

Grefenstette J 1986 Optimization of control parameters for genetic algorithms *IEEE Trans. Syst. Man Cybernet.* **SMC-16** 122–8

——1995 *Virtual Genetic Algorithms: First Results* Navy Center for Applied Research in AI Report AIC-95-013

Grefenstette J, Ramsey C and Schultz A 1990 Learning sequential decision rules using simulation models and competition *Machine Learning* **54** 355–81

Holland J 1975 *Adaptation in Natural and Artificial Systems* (Ann Arbor, MI: University of Michigan Press)

Hong I, Kahng A and Moon B 1995 Exploiting synergies of multiple crossovers: initial studies *Proc. IEEE Int. Conf. on Evolutionary Computation*

Julstrom B 1995 What have you done for me lately? adapting operator probabilities in a steady-state genetic algorithm *Proc. 6th Int. Conf. on Genetic Algorithms (Pittsburgh, PA, 1995)* ed L J Eshelman (San Mateo, CA: Morgan Kaufmann) pp 81–7

Kahng A and Moon B 1995 Towards more powerful recombinations *Proc. 6th Int. Conf. on Genetic Algorithms (Pittsburgh, PA, 1995)* ed L J Eshelman (San Mateo, CA: Morgan Kaufmann) pp 96–103

Koza J 1994 *Genetic Programming II: Automatic Discovery of Reusable Subprograms* (Cambridge, MA: MIT Press)

Lee M and Takagi H 1993 Dynamic control of genetic algorithms using fuzzy logic techniques *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 77–83

Levenick J 1995 Metabits: generic endogenous crossover control *Proc. 6th Int. Conf. on Genetic Algorithms (Pittsburgh, PA, 1995)* ed L J Eshelman (San Mateo, CA: Morgan Kaufmann) pp 88–95

Manderick B, de Weger M and Spiessens P 1991 The genetic algorithms and the structure of the fitness landscape *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, 1991)* ed R K Belew and L B Booker (San Mateo, CA: Morgan Kaufmann) pp 143–50

Moon B and Bui T 1994 Analyzing hyperplane synthesis in genetic algorithms using clustered schemata *Parallel Problem Solving from Nature—III (Lecture Notes in Computer Science 806)* pp 108–18

Oliver I, Smith D and Holland J 1987 A study of permutation crossover operators on the traveling salesman problem *Proc. 2nd Int. Conf. on Genetic Algorithms (Pittsburgh, PA, 1987)* ed J J Grefenstette (Hillsdale, NJ: Erlbaum) pp 224–30

Peck C and Dhawan A 1995 Genetic algorithms as global random search methods: an alternative perspective *Evolutionary Computation* (Cambridge, MA: MIT Press) **3 1** 39–80

Radcliffe N 1991 Forma analysis and random respectful recombination *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, 1991)* ed R K Belew and L B Booker (San Mateo, CA: Morgan Kaufmann) pp 222–9

——1994 Fitness variance of formae and performance prediction *Proc. 3rd Foundations of Genetic Algorithms Workshop* ed M Vose and D Whitley (San Mateo, CA: Morgan Kaufmann) pp 51–72

Rosca J 1995 Genetic programming exploratory power and the discovery of functions *Proc. 4th Annu. Conf. on Evolutionary Programming (San Diego, CA, 1995)* ed J R McDonnell, R G Reynolds and D B Fogel (Cambridge, MA: MIT Press) pp 719–36

Rosenberg R 1967 *Simulation of Genetic Populations with Biochemical Properties* PhD Dissertation, University of Michigan

Schaffer J, Caruana R, Eshelman L and Das R 1989 A study of control parameters affecting on-line performance of genetic algorithms for function optimization *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 51–60

Schaffer J and Eshelman K 1991 On crossover as an evolutionarily viable strategy *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, 1991)* ed R K Belew and L B Booker (San Mateo, CA: Morgan Kaufmann) pp 61–8

Schaffer J and Morishima A 1987 An adaptive crossover distribution mechanism for genetic algorithms *Proc. 2nd Int. Conf. on Genetic Algorithms (Pittsburgh, PA, 1987)* ed J J Grefenstette (Hillsdale, NJ: Erlbaum) pp 36–40

Schwefel H-P 1995 *Evolution and Optimum Seeking* (New York: Wiley)

Smith S 1980 Flexible learning of problem solving heuristics through adaptive search *Proc. 8th Int. Conf. on Artificial Intelligence* pp 422–5

Spears W 1992 Crossover or Mutation? *Proc. 2nd Foundations of Genetic Algorithms Workshop* ed D Whitley (San Mateo, CA: Morgan Kaufmann) pp 221–37

——1995 Adapting crossover in evolutionary algorithms *Proc. 4th Ann. Conf. on Evolutionary Programming (San Diego, CA, 1995)* ed J R McDonnell, R G Reynolds and D B Fogel (Cambridge, MA: MIT Press) pp 367–84

Spears W and De Jong K 1991 On the virtues of parameterized uniform crossover *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, 1991)* ed R K Belew and L B Booker (San Mateo, CA: Morgan Kaufmann) pp 230–6

Srinivas M and Patnaik L 1994 Adaptive probabilities of crossover and mutation in genetic algorithms *IEEE Trans. Syst. Man Cybernet.* **SMC-244** 656–67

Starkweather T, McDaniel S, Mathias K, Whitley D and Whitley C 1991 A comparison of genetic sequencing operators *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, 1991)* ed R K Belew and L B Booker (San Mateo, CA: Morgan Kaufmann) pp 69–76

Syswerda G 1989 Uniform crossover in genetic algorithms *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 2–9

——1992 Simulated crossover in genetic algorithms *Proc. 2nd Foundations of Genetic Algorithms Workshop* ed D Whitley (San Mateo, CA: Morgan Kaufmann) pp 239–55

White T and Oppacher F 1994 Adaptive crossover using automata *Proc. Parallel Problem Solving from Nature Conf.* ed Y Davidor, H-P Schwefel and R Männer (New York: Springer)

Wilson S 1986 *Classifier System Learning of a Boolean Function* Rowland Institute for Science Research Memo RIS-27r

Wright A 1991 Genetic algorithms for real parameter optimization *Proc. Foundations of Genetic Algorithms Workshop* ed G Rawlins (San Mateo, CA: Morgan Kaufmann) pp 205–18

E2.1 Efficient implementation of algorithms

John Grefenstette

Abstract

This section discusses techniques for the efficient implementation of evolutionary algorithms, including generating random numbers, implementing the genetic operators, and reducing computational effort in the evaluation phase.

E2.1.1 Introduction

The implementation of evolutionary algorithms requires the usual attention to software engineering principles and programming techniques. Given the variety of evolutionary techniques presented in this handbook, it is not possible to present a complete discussion of implementation details. Consequently, this section focuses on a few areas that may substantially contribute to the overall efficiency of the implementation. These are (i) random number generators, (ii) genetic operators, (iii) selection, and (iv) the evaluation phase.

E2.1.2 Random number generators

The subject of random number generation has a very extensive literature (Knuth 1969), so this section provides only a brief introduction to the topic. In fact, most programming language libraries include at least one random number generator, so it may not be necessary to implement one as part of an evolutionary algorithm. However, it is important to be aware of the properties of the random number generator being used, and to avoid the use of a poorly designed generator. For example, Booker (1987) points out that care must be taken when using simple multiplicative random number generators to initialize a population, because the values that are generated may not be randomly distributed in more than one dimension. Booker recommends generating random populations as usual, and then performing repeated crossover operations with uniform random pairing. Ideally, this would be done to the point of stochastic equilibrium, meaning that the probability of occurrence of every schema is equal to the product of the proportions of its defining alleles.

One commonly used method of generating a pseudorandom sequence is the *linear congruential method*, defined by the relation

$$X_{n+1} = (a_r X_n + c_r) \bmod m_r \qquad n \geq 0$$

where $m_r > 0$ is the *modulus*, X_0 is the starting value (or *seed*), and a_r and c_r are constants in the range $[0, m_r)$. The properties of the linear congruential method depend on the choices made for the constants a_r, c_r, and m_r. (See the book by Knuth (1969) for a thorough discussion.) Reasonably good results can be obtained with the following values on a 32-bit computer (Press *et al* 1988):

$$a_r = 4096 \qquad c_r = 150\,889 \qquad m_r = 714\,025.$$

The following routine uses the linear congruential method to generate a uniformly distributed random number in the range $[0, 1)$:

Input: the current random seed, $Seed$.
Output: u_r, a uniformly distributed random number in the range $[0, 1)$; the seed is updated as a side-effect.

```
1   Urand(Seed):
2       Seed ← (a_r Seed + c_r) mod m_r;
3       u_r ← Seed/m_r;
4       return u_r ;
```

Given $Urand$ above, the following generates a uniformly distributed real value in the range $[a, b)$:

Input: lower bound a, upper bound b.
Output: u, a uniformly distributed random number in the range $[a, b)$.

```
1   U(a, b):
2       u_r ← Urand(Seed);
3       u ← a + (b − a) u_r;
4       return u ;
```

Note that the above procedure always returns a value strictly less than the upper bound b. The following generates an integer value from the range $[a, b]$ (inclusive of both endpoints):

Input: lower bound a (an integer), upper bound b (an integer).
Output: i, a uniformly distributed random integer in the range $[a, b]$.

```
1   Irand(a, b):
2       u_r ← Urand(Seed);
3       i ← a + ⌊(b − a + 1) u_r⌋;
4       return i ;
```

It is often required to generate variates from a normal distribution in evolutionary algorithms; for example, the mutation perturbations in *evolution strategies* (ESs) and *evolutionary programming* (EP) may be specified as normally distributed random variables. One way to implement an approximately normal distribution is based on the central-limit theorem, which states that, as $n \to \infty$, the sum of n independent, identically distributed (IID) random variables has approximately a normal distribution with a mean of $n\zeta$ and a variance of $n\sigma^2$, where ζ and σ^2 are the mean and variance, respectively, of the IID random variables. If the IID random variables follow the standard uniform distribution, as computed by $U(0, 1)$ above for example, then $\zeta = 1/2$ and $\sigma^2 = 1/12$. It follows that summing n samples from $U(0, 1)$ gives an approximation to a sample from a normal distribution with mean $n/2$ and variance $n/12$. The following illustrates this approach:

B1.3, B1.4

Input: mean ζ, standard deviation σ.
Output: x, a variate from the normal distribution with mean ζ and standard deviation σ.

```
1   N(ζ, σ):
2       sum ← 0;
        {n is a user-selected constant, n ≥ 12}
3       for i ← 1 to n do
4           sum ← sum + U(0, 1);
        od
5       z ← sum − n/2;
6       z ← z/(n/12);
        {z now approximates a sample from the standard normal distribution}
7       x ← ζ + σz;
8       return x ;
```

Studies have shown that fairly good results can be obtained with $n = 12$, thus eliminating the need for the division operation in the computation of the variance in line 6 (Graybeal and Pooch 1980).

Finally, evolutionary algorithms often require the generation of a randomized shuffle or permutation of objects. For example, it may be desired to shuffle a population before performing pairwise crossover. The following code implements a random shuffle:

Input: *perm*, an integer array of size n.
Output: *perm*, an array containing a random permutation of values from 1 to n.

```
1   Shuffle(perm):
2       for i ← 1 to n do
3           perm[i] ← i;
        od
4       for i ← 1 to n − 1 do
5           j ← Irand(i, n);
6           {swap items i and j}
7           temp ← perm[i] ;
8           perm[i] ← perm[j] ;
9           perm[j] ← temp;
        od
10      return perm ;
```

E2.1.3 The selection operator

As discussed in Chapter C2, there is a wide variety of selection schemes for evolutionary algorithms. C2 However, many selection algorithms involve two fundamental steps:

(i) Compute selection probabilities for the current population based on fitness.
(ii) Sample the current population based on the selection probabilities to obtain clones which may then be subject to mutation or recombination.

Section C2.2.3 discusses efficient techniques for the second step, the sampling of the current population C2.2.3 according to the selection probabilities. The SUS algorithm (Baker 1987) assigns a number of offspring to each individual in the population, based on the selection probability distribution. SUS is simple to implement and is optimally efficient, making a single pass over the individuals to assign all offspring.

E2.1.4 Crossover operators

As discussed in Section C3.3, many crossover operators have been proposed and implemented for C3.3 evolutionary algorithms. The reader is referred to the parts of that section for specification and implementation instructions.

E2.1.5 The mutation operator

In contrast to ESs and EP, *genetic algorithms* usually apply mutation at a uniform rate (p_{m}) across all B1.2 genes and across all individuals in the population. After the new population has been selected, each gene position is subjected to a Bernoulli trial, with a probability of success given by the mutation rate parameter p_{m}. The most obvious implementation involves sampling from a random variable for each gene position. However, if $p_{\mathrm{m}} \ll 1$, the computational cost can be substantially reduced by avoiding the gene-by-gene calls to the random number generator. A sequence of Bernoulli trials with probability p_{m} has an interarrival time that follows a geometric distribution. A sample from such a geometric distribution, representing the number of gene positions until the next mutation, can be computed as follows (Knuth 1969, p 131):

$$m_{\mathrm{next}} = \left\lceil \frac{\ln u}{\ln(1 - p_{\mathrm{m}})} \right\rceil$$

where u is a sample from a random variable uniformly distributed over $(0, 1)$. Assuming that the variable m_{next} is initialized according to this formula, the following pseudocode illustrates the mutation procedure for an individual:

Input: an individual $\boldsymbol{a}$ of length n; a mutation probability, p_{m}; the next arrival point for mutation, m_{next}.
Output: the mutated individual a_i, and the updated next mutation location, p_{m}.

```
1   mutate-individual(a, p_m, m_next):
        {Note: n is the number of genes in an individual}
2       while m_next < n do
            {mutate-gene is a function that alters the individual a at position m_next, }
            {e.g. flip the bit at position m_next or sample from a distribution}
            {using σ_{m_next}, the standard deviation for this position.}
3           mutate-gene(a, m_next);
            {compute new interarrival interval}
4           if (p_m = 1)
            then
5               m_next ← m_next + 1;
            else
6               sample u ~ U(0, 1);
7               m_next ← m_next + ⌈ ln u / ln(1 − p_m) ⌉;
            fi
        od
        {prepare for next individual, essentially treating}
        {the entire population as a single string of gene positions}
8       m_next ← m_next − n;
9       return a, m_next ;
```

E2.1.6 The evaluation phase

In practice, the time required to evaluate the individuals usually dominates the rest of the computational effort of an evolutionary algorithm. This is especially true if the computation of the objective function requires running a substantial program, such as a simulation of a complex system. It follows that very often the most significant way to speed up an evolutionary algorithm is to reduce the time spent in the evaluation phase. This section considers three ways to reduce evaluation time:

(i) avoid redundant evaluations;
(ii) use sampling to perform approximate evaluations; and
(iii) exploit parallel processing.

E2.1.6.1 Deterministic evaluations

The objective function may be deterministic (e.g. the tour length in a *traveling salesman problem*) or nondeterministic (e.g. the outcome of a stochastic simulation). If the objective function is deterministic, then it may be worthwhile to avoid evaluating the same individual twice. This can be implemented as follows: G9.5

(i) When creating an offspring clone from a parent, copy the parent's objective function value into the offspring. Mark the offspring as *evaluated.*
(ii) After the application of a genetic operator, such as mutation or crossover, to an individual, check to see whether the resulting individual is different from the original one. If so, mark the individual as *unevaluated*.
(iii) During the evaluation phase, only evaluate individuals that are marked *unevaluated.*

The cost of this extra processing is dominated by step (ii), which can be accomplished in at most $O(n)$ steps for individuals of length n. The cost may be a constant for some operators (e.g. mutation). In many cases, this extra processing will be worthwhile to avoid the cost of an additional evaluation.

For some applications, it may be worthwhile to cache every individual along with its associated evaluation, so that the same individual is never evaluated twice during the course of the evolutionary algorithm. This method clearly involves significant additional overhead in terms of storage space and in

matching each newly generated individual against the previously generated individuals, so its use should be reserved for applications with very expensive, but deterministic, objective functions.

E2.1.6.2 Monte Carlo evaluation

If the objective function is nondeterministic, then each individual should be reevaluated during each generation. One important special class of nondeterministic objective functions is those computed through Monte Carlo procedures, in which a number of random samples are drawn from a distribution and the objective function returns the average of the sample values. If the objective function involves Monte Carlo sampling, then the user must determine how much effort should be expended per evaluation.

Fitzpatrick and Grefenstette (1988) discuss the case of a generational genetic algorithm using *proportional selection*, in which the evaluation of individuals is performed by a Monte Carlo procedure that iterates a fixed number of times, s, for each evaluation. It is shown that as the number of samples s is decreased (to save evaluation time), the accuracy of the estimation of a schema's fitness decreases much more slowly than the accuracy of the observed fitness of the individuals in the population. Assuming that the quality of the search performed by a genetic algorithm depends on the quality of its estimates of the performance of schemas, this suggests that genetic algorithms can be expected to perform well even using relatively small values for s. This analysis also suggests that the estimate of the average performance of the hyperplanes present in a given population may be improved by trading off an increase in the population size (thereby testing a greater number of representatives from each hyperplane) with a corresponding decrease in the number s of Monte Carlo samples per evaluation. The effect of this tradeoff on the overall runtime depends on the ratio of the evaluation costs to the other overhead associated with the genetic algorithm. A similar study by Hammel and Bäck (1994) showed a similar result, that ESs are also robust in the face of a noisy evaluation function. However, this study also showed that increasing the population size yielded a smaller performance improvement than increasing the sampling rate s for ESs.

C2.2

E2.1.6.3 Parallel evaluation

Since evolutionary algorithms are characterized by their use of a population, it is natural to view them as parallel algorithms. In generational evolutionary algorithms, substantial savings in elapsed time can often be obtained by performing evaluations in parallel. In the simplest form of parallelism, a master process performs all the function of the evolutionary algorithm except evaluation of individuals, which are performed in parallel by worker processes operating on separate processors. The master process waits for all workers to return the evaluated individuals before varying on with the next generation.

One definition of speedup for parallel algorithms is

$$S(p, N) = \frac{T(1, N)}{T(p, N)}$$

where $T(p, N)$ is the time required to perform a task of size N on p processors. The parallel evolutionary algorithm described above is a form of *parallel generate-and-test* algorithm, in which N possible solutions are generated on each iteration (i.e. in this context $N = \mu$, the population size). The efficiency of a parallel generate-and-test algorithm depends on ensuring that the workers finish their assigned tasks at nearly the same time. Since the master waits for all workers to complete, the time between the completion of a given worker and the completion of all others workers is wasted. For a such an algorithm, the maximum speedup with p processors is given by

$$S(p, N) = \frac{T(1, N)}{T(p, N)} = \frac{\alpha + \beta N}{\alpha + (\beta N/p)}$$

where α is the time required by the master process to generate the candidate solutions, and β is the time required to evaluate a single solution. Speedup approaches the ideal value of p as α/β approaches zero (Grefenstette 1995). In practice, the speedup is usually less than p due to communication overhead. Further degradation occurs if N is not evenly divided by p, since some workers will have more tasks to perform than others.

Other forms of parallel processing are discussed in Sections C6.3 and C6.4.

C6.3, C6.4

References

Baker J E 1987 Reducing bias and inefficiency in the selection algorithm *Proc. 2nd Int. Conf. on Genetic Algorithms (Cambridge, MA, 1987)* ed J J Grefenstette (Hillsdale, NJ: Erlbaum) pp 14–21

Booker L B 1987 Improving search in genetic algorithms *Genetic Algorithms and Simulated Annealing* ed L Davis (San Mateo, CA: Morgan Kaufmann)

Fitzpatrick J M and Grefenstette J J 1988 Genetic algorithms in noisy environments *Machine Learning* **3** 101–20

Graybeal W J and Pooch U W 1980 *Simulation: Principles and Methods* (Cambridge, MA: Withrop)

Grefenstette J J 1995 Robot learning with parallel genetic algorithms on networked computers *Proc. 1995 Summer Computer Simulation Conf. (SCSC '95)* ed T Oren and L Birta (Ottawa: The Society for Computer Simulation) pp 352–57

Hammel U and Bäck T 1994 Evolution strategies on noisy functions: how to improve convergence properties *Parallel Problem solving from Nature—PPSN III (Proc. Int. Conf. on Evolutionary Computation and 3rd Conf. on Parallel Problem Solving from Nature, Jerusalem, October 1994)* (*Lecture Notes in Computer Science 866*) ed Yu Davidor, H-P Schwefel and R Männer (Berlin: Springer) pp 159–68

Knuth D E 1969 *Seminumerical Algorithms: the Art of Computer Programming* vol 2 (Reading, MA: Addison-Wesley)

Press W M, Flannery B P, Teukolsky S A and Vetterling W T 1988 *Numerical Recipes in C: the Art of Scientific Computing* (Cambridge: Cambridge University Press)

E2.2 Computation time of evolutionary operators

Günter Rudolph and Jörg Ziegenhirt

Abstract

The runtime of one iteration of an evolutionary algorithm depends on several factors: the population size, the problem dimension, the required time to calculate an objective function value, and the computation time for each evolutionary operator. These operators may be divided into selection, mutation, and recombination operators. Since there is a huge variety of evolutionary operators, only the most common ones are investigated with regard to their computation time.

E2.2.1 Asymptotical notations

The computation time of the evolutionary operators will be presented in terms of asymptotics. The usual notation in this context is summarized below. Further information can be found, for example, in the book by Horowitz and Sahni (1978).

Let f and g be real-valued functions with domain $\mathbb{N}$.

(i) $f(n) = O(g(n))$ if there exist constants $c, n_0 > 0$ such that $|f(n)| \leq c\,|g(n)|$ for all $n \geq n_0$.
(ii) $f(n) = \Omega(g(n))$ if there exist constants $c, n_0 > 0$ such that $|g(n)| \leq c\,|f(n)|$ for all $n \geq n_0$.
(iii) $f(n) = \Theta(g(n))$ if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.
(iv) $f(n) = o(g(n))$ if $f(n)/g(n) \to 0$ as $n \to \infty$.

E2.2.2 Computation time of selection operators

Suppose that the selection operator chooses μ individuals from λ individuals with $\mu \leq \lambda$. Thus, the input to the selection procedure consists of λ values $(v_1, \ldots, v_\lambda)$ usually representing the fitness values of the individuals. The output is an array of indices referring to the selected individuals. We shall assume that the fitness is to be maximized.

E2.2.2.1 Proportional selection via roulette wheel

For *proportional selection* via the roulette wheel it is necessary to assume positive values v_k. At first we cumulate the values v_k such that $c_k = \sum_{i=1}^{k} v_i$ for $k = 1, \ldots, \lambda$. This requires $\mathrm{O}(\lambda)$ time. The next step is repeated μ times: draw a uniformly distributed random number U from the range $(0, c_\lambda) \subset \mathbb{R}$. Since the values c_k are sorted by construction, binary search takes $\mathrm{O}(\log \lambda)$ time to determine the index i with $c_i = \max\{k \in K : U < c_k\}$ where $K = \{1, \ldots, \lambda\}$. Consequently, the computation time is $\mathrm{O}(\lambda + \mu \log \lambda)$ and $\mathrm{O}(\lambda \log \lambda)$ if $\mu = \lambda$. C2.2

E2.2.2.2 Stochastic universal sampling

C2.2

The algorithm given by Baker (1987) is an almost deterministic variant of proportional selection as described above and requires $\mathrm{O}(\lambda)$ time if $\mu = \lambda$.

E2.2.2.3 q-ary tournament selection C2.3

Choose q indices $\{i_1, \ldots, i_q\}$ from $\{1, \ldots, \lambda\}$ at random and determine the index k with $v_k \geq v_i$ for all $i \in \{i_1, \ldots, i_q\}$. This is done in $O(q)$ time. Consequently, the μ-fold repetition of this operation requires $O(q\mu)$ time.

E2.2.2.4 (μ, λ) selection C2.4.4

This type of selection can be done in $O(\lambda)$ time although all public domain evolutionary algorithm (EA) software packages we are aware of employ algorithms with worse worst-case running time. Moreover, it should be noted that this selection method can be interleaved with the process of generating new individuals. This reduces the memory requirements from λ to μ times an individual's size, which can be important when the individuals are very large objects or when computer memory is small. The FORTRAN code originating from 1975 (when memory was small) that comes with the disk of Schwefel (1995) realizes the following method. The first μ individuals that are generated are stored in an array and the worst one is determined. This requires $O(\mu)$ time. Each further individual is checked to establish whether its fitness value is better than the worst one in the array. If so, the individual in the array is replaced by the new one and the worst individual in the modified array is determined in $O(\mu)$ time. This can happen $\lambda - \mu$ times such that the worst-case runtime is $O(\mu + (\lambda - \mu)\,\mu)$. However there is a better algorithm that is based on *Heapsort*. After the first μ individuals have been stored in the array, the array is rearranged as a heap requiring $O(\mu)$ time. The worst individual in the heap is known by nature of the heap data structure. If a better individual is generated, the worst individual in the heap is replaced by the new one and the heap property is repaired which can be done in $O(\log \mu)$ time. Thus, the worst-case runtime improves to $O(\mu + (\lambda - \mu)\,\log \mu)$. For further details on heapsort or the heap data structure see for example the book by Sedgewick (1988).

Now we assume that λ individuals have been generated and the fitness values are stored in an array. The naive approach to select the μ best ones is to sort the array such that the μ best individuals can be extracted easily. This can be done in $O(\lambda \log \lambda)$ time by using, for example, heapsort. Again, there are better methods. Note that it is not necessary to sort the array completely—it suffices to create a heap in $O(\lambda)$ time, to select the best element, and to apply the heap repairing mechanism to the remaining elements. Since this must be done only μ times and since the repairing step requires $O(\log \lambda)$ time, the entire runtime is bounded by $O(\lambda + \mu \log \lambda)$. Other sorting algorithms can be modified for this purpose as well: a variant of *quicksort* is given by Press *et al* (1986), whereas Fischetti and Martello (1988) present a sophisticated quicksort-based method that extracts the μ best elements in $O(\lambda)$ time. Since the method is not very basic (and the FORTRAN code is optimized by hand!) we refrain from presenting a description here.

Table E2.2.1. Time and memory requirements of several methods to realize (μ, λ) selection.

No	Method	Worst-case runtime	Required memory
1	Schwefel (1975)	$O(\mu + (\lambda - \mu)\,\mu)$	$O(\mu)$
2	Schwefel + heap	$O(\mu + (\lambda - \mu)\,\log \mu)$	$O(\mu)$
3	Modified heapsort	$O(\lambda + \mu \log \lambda)$	$O(\lambda)$
4	Modified quicksort	$O(\mu\lambda)$	$O(\lambda)$
5	Fischetti–Martello	$O(\lambda)$	$O(\lambda)$

A summary of time and place requirements of the methods to realize (μ, λ) selection is given in table E2.2.1. We have made extensive tests to identify the break-even points of the above algorithms. Under the assumption that the fitness values in the array are arranged randomly and do not have a partial preordering, it turned out that up to $\lambda = 100$ and $\mu = 30$ all methods worked equally well. For larger $\lambda \in \{200, \ldots, 1000\}$ and $\mu/\lambda > 0.3$ method 2 clearly outperforms method 1, whereas methods 3–5 with $O(\lambda)$ place requirements do not reveal significant differences in performance, although method 4 has a trend to be slightly worse. Moreover, methods 2 and 4 perform similarly up to $\lambda = 1000$.

E2.2.2.5 $(\mu + \lambda)$ selection C2.4.4

This type of selection can be realized similarly to (μ, λ) selection. If the generating and selecting process is interleaved, we need $O(\mu)$ place and $O(\lambda \log \mu)$ time. Assume that the old population of μ individuals is arranged in heap order and that a better individual was generated. Then only $O(\log \mu)$ time is necessary to replace the worst individual in the heap by the new one and to repair the heap, which can happen at most λ times. After all λ individuals are processed, the new population is of course in heap order. Therefore, once the initial population is arranged as a heap the population will remain in heap order forever.

If the λ individuals are generated before selection begins, we need $O(\mu + \lambda)$ place and $O(\mu + \lambda)$ time: simply run the algorithm of Fischetti–Martello on the array of size $\mu + \lambda$.

E2.2.2.6 q-fold binary tournament selection (EP selection) C2.6

Similar to the selection methods given in sections E2.2.2.1–3, this type of selection cannot be performed in an interleaved manner. The description given here follows Fogel (1995), p 137. Suppose that the λ individuals have been generated such that there is an array of $\mu + \lambda$ individuals. For each individual draw q individuals at random and determine the number of times the individual is better than the q randomly chosen ones. This number in the range 0–q is the score of the individual. Thus, the scores of all individuals can be obtained in $O((\mu + \lambda)\, q)$ time. Finally, the μ individuals with highest score are selected by the Fischetti–Martello algorithm in $O(\mu + \lambda)$ time. Alternatively, since the score can only attain $q + 1$ different values, the μ individuals with highest scores could be selected via bucket sort in $O(\mu + \lambda)$ time. Altogether, the runtime is bounded by $O((\mu + \lambda)\, q)$.

E2.2.3 Computation time of mutation operators

We assume that the mutation operator is applied to an n-tuple and that this operation is an in-place operation. At first we presuppose that an elementary mutation of one component of the tuple can be done in constant time.

Let $p_i \in (0, 1\,]$ be the probability that component $i = 1, \ldots, n$ will undergo an elementary mutation. Then mutation works as follows: for each i draw a uniformly distributed random number $u \in (0, 1)$ and mutate component i if $u \leq p$, otherwise leave it unaltered. Evidently, this requires $\Theta(n)$ time. If $p_i = 1$ for all $i = 1, \ldots, n$ we can of course refrain from drawing n random numbers. Although this does not decrease the asymptotic runtime, there will be a saving of real time. These savings can accumulate to a considerable amount of real time. Therefore, even apparently very simple operations deserve a careful consideration. For example, let $p_i = p \ll 1$ for all $i = 1, \ldots, n$ and let B be the number of components that will be affected by mutation. Since B is a binomially distributed random variable with expectation np, the average number of elementary mutations will reduce to 1 if $p = 1/n$. This can be realized by a simulation of the original mutation process. Imagine a concatenation of all n-tuples of a population of size λ such that we obtain a (λn)-tuple for each generation. If the EA is stopped after N generations the concatenation of all (λn)-tuples yields one large $(N\lambda n)$-tuple. Let U_k be a sequence of independent uniformly distributed random variables over $[\,0, 1)$. Then the random variable $M_k = 1_{[\,0,p)}(U_k)$ indicates whether component k of the $(N\lambda n)$-tuple has been mutated ($M_k = 1$) or not ($M_k = 0$). Let $T = \min\{k \geq 1 : M_k = 1\}$ be the random time of the first elementary mutation. Note that T has a geometrical distribution with probability distribution function $\mathsf{P}\{T = k\} = p\,(1 - p)^{k-1}$ and expectation $\mathsf{E}[T] = 1/p$. Since geometrical random numbers can be generated via

$$T = 1 + \left\lfloor \frac{\log(1 - U)}{\log(1 - p)} \right\rfloor$$

where U is a uniformly distributed random number over $[\,0, 1)$, we can simulate the original mutation process by drawing geometrical random numbers. Let T_ν denote the νth outcome of random variable T. Then the values of the partial sums of the series

$$\sum_{\nu=1}^{\infty} T_\nu$$

are just the indices of the $(N\lambda n)$-tuple at which elementary mutations occur. An implementation of this method (by drawing T on demand) yields a theoretical average speedup of $1/p$, but since the generation

of a geometrical random number requires the logarithm function the practical average speedup is slightly smaller.

The initial assumption that elementary mutations require constant time is not always appropriate. For example, let $\boldsymbol{x} \in X^n = \mathbb{R}^n$ and let $\boldsymbol{z} \sim N(0, C)$ be a normally distributed random vector with zero mean and positive definite, symmetric covariance matrix C. Unless C is a diagonal matrix the components of the random vector $\boldsymbol{z}$ are correlated. Since we need $\mathrm{O}(n)$ standard normally distributed random numbers and $\mathrm{O}(n^2)$ elementary operations to build random vector $\boldsymbol{z}$, the entire mutation operation $\boldsymbol{x} + \boldsymbol{z}$ requires $\mathrm{O}(n^2)$ time; consequently, an elementary mutation operation requires $\mathrm{O}(n)$ time.

E2.2.4 Computation time of recombination operators

Assume that the input to *recombination* consists of $\rho \in \{2, \ldots, \mu\}$ n-tuples whereas the output is a single n-tuple. Consequently, for every recombination operator we have the bound $\Omega(n)$. Thus, the runtime for one-point crossover, uniform crossover, intermediate recombination, and gene pool recombination is $\Theta(n)$ whereas k-point crossover requires $\mathrm{O}(n + k \log k)$ time. C3.3

Usual implementations of k-point crossover do not demand that the k crossover points are pairwise distinct. Therefore, we may draw k random numbers from the range 1 to $n - 1$ and sort them. These numbers are taken as the positions to swap between the tuples.

E2.2.5 Final remarks

Without any doubt, it is always useful to employ the most efficient data structures and algorithms to realize variation and selection operators, but in almost all practical applications most time is spent during the calculation of the objective function value. Therefore, the realization of this operation ought to be always checked with regard to potential savings of computing time.

References

Baker J 1987 Reducing bias and inefficiency in the selection algorithm *Proc. 2nd Int. Conf. on Genetic Algorithms and their Applications (Pittsburg, PA, July, 1987)* ed Grefenstette J (Hillsdale, NJ: Erlbaum) pp 12–21

Fischetti M and Martello S 1988 A hybrid algorithm for finding the kth smallest of n elements in $\mathrm{O}(n)$ time *Ann. Operations Res.* **13** 401–19

Fogel D B 1995 *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence* (New York: IEEE)

Horowitz E and Sahni S 1978 *Fundamentals of Computer Algorithms* (London: Pitman)

Schwefel H-P 1995 *Evolution and Optimum Seeking* (New York: Wiley)

Sedgewick R 1988 *Algorithms* 2nd edn (Reading, MA: Addison-Wesley)

Press W H, Flannery B P, Teukolsky S A and Vetterling W T 1986 *Numerical Recipes* (Cambridge: Cambridge University Press)

E2.3 Hardware realizations of evolutionary algorithms

Tetsuya Higuchi and Bernard Manderick

Abstract

An overview of existing hardware and parallel implementations of evolutionary algorithms is given. Also, evolvable hardware where the evolutionary algorithm operates at the hardware level is discussed.

E2.3.1 Introduction

In order to use evolutionary algorithms (EAs) including *genetic algorithms* (GAs) in real time or for hard real-world applications, their current speed has to be increased several orders of magnitude. B1.2

This section reviews research activities related to hardware realizations of EAs. First, we consider parallel implementations of GAs on different parallel machines. Then, we focus on more dedicated hardware systems for EAs. For example, a TSP GA machine, a wafer-scale GA machine, and vector processing of GA operators are described. Here, we discuss also a new research field, called *evolvable hardware* (EHW), since it has close relationships with hardware realizations of EAs.

This section is organized as follows. First, we discuss different PGAs. Second, we review dedicated hardware systems for EAs, and third, we take a closer look at EHW.

E2.3.2 Parallel genetic algorithms

Parallel GAs (PGAs) can be classified along two dimensions. The first one is the parallel programming paradigm and the related computer architecture on which they are running. The second one is the structure of the population used. These two dimensions are not orthogonal: some population structures are better suited for certain architectures.

In the next three subsections, we discuss the different architectures and the different population structures and we classify PGAs according to these two dimensions.

E2.3.2.1 Parallel computer architectures

Basically, there are two approaches to parallel programming. In *control level parallelism*, one tries to identify parts of an algorithm that can operate independently of each other and can therefore be executed in parallel. The main problem with this approach is to identify and synchronize these independent parts of the algorithm. Consequently, control level parallelism is limited in the number of parallel processes that can be coordinated. In practice, this limit lies at the order of ten.

In the second approach, *data level parallelism*, one tries to identify independent data elements that can be processed in parallel, but with the same instructions. It is clear that this approach works best on problems with large numbers of data. For these problems, data level parallelism is ideally suited to program *massively* parallel computers. Consequently, this approach makes it possible to fully exploit the power offered by parallel systems.

These parallel algorithms have to be implemented on one of the existing parallel architectures. A common classification of these is based on how they handle instruction and data streams. The two main classes are the *single instruction, multiple data* (SIMD) and the *multiple instruction, multiple data* (MIMD). An SIMD machine is executing a single instruction stream acting upon many data streams at the same time.

Its advantage is that it is easily programmed. In contrast, an MIMD machine has multiple processors, each one executing independently its own instructing stream operating on its own data stream. MIMD computers can be divided into *shared-memory MIMD* and *distributed-memory MIMD* machines. They differ in the way the individual processors communicate. The processors in a shared-memory system communicate by reading from and writing to memory locations in a common address space. Since only one processor can have access to a shared memory location at a given time, this limits parallelization. Therefore, shared-memory systems are suited for control level parallelism but not for data level parallelism. However, these systems can be programmed easily. In a distributed-memory MIMD machine each processor has its own local memory. Communication between processors proceeds through passing data over a communication network. Many different network organizations are possible. The big advantage of distributed-memory MIMD machines is that they can be scaled to include a virtually unlimited number of processors without degrading performance. They are suited for both control level and data level parallelism. However, they are much more difficult to program.

E2.3.2.2 Population structure in nature

Since the very beginning of population genetics, the impact of the population structure on evolution has been stressed: the way how a population is structured influences the evolutionary process (Wright 1931). A number of population structures have been introduced and investigated theoretically. We discuss them briefly. More details can be found elsewhere in this handbook.

The importance of this work for GAs is that it shows that PGAs are fundamentally different from the standard GA and that they different from each other depending on the population structure used.

According to Fisher, populations are effectively *panmictic* (Fisher 1930). This is, all individuals compete with each other during the selection process, and every individual can potentially mate with every other one. The standard GA has a panmictic population structure.

The *island* and the *stepping stone* population structures are closely related. In both cases, the population is divided into a number of *demes*, which are semi-independent subpopulations that remain loosely coupled to neighboring demes by migrants. C6.3

In general, the island models are characterized by relatively large demes, with all-to-all migration patterns between them. The stepping stone models are characterized by smaller demes arranged in a lattice, with migration patterns between nearest neighbors only.

Finally, in the *isolation-by-distance* model, the population is spread across a continuum. Each individual interacts only with individuals in its immediate neighborhood. Each small neighborhood is like a deme except that now the demes are overlapping. Individuals in a deme are implicitly isolated instead of explicitly as is the case in the two previous population structures.

E2.3.2.3 An overview of parallel genetic algorithms

In this section, we give an overview of existing PGAs. For each PGA we discuss the *population structure* used, the amount of *parallelism*, and its *scalability*. Parallelism is measured by counting the number of individuals that can be treated in parallel during each step of the GA. Scalability reflects how an increase in population size affects the total execution time. These measures represent the two most important benefits of PGAs: the increased speed of execution and the possibility to work with large populations.

In *coarse-grained PGAs* the population is structured as in the stepping stone population model. A large population is divided into a number of equally sized subpopulations. The parallelism is obtained by the parallel execution of a number of classic GAs, each of which operates on one of the subpopulations. Occasionally, the parallel processes communicate to exchange migrating individuals. Because each process consists of running a complete GA, it is rather difficult to synchronize all processes. Consequently, coarse-grained PGAs are most efficiently implemented on MIMD machines. In particular, they are suited for implementation on distributed-memory MIMD systems where they can take full advantage of the virtually unlimited number of processors.

In coarse-grained PGAs, the parallelism is limited because each step still operates on a (sub)population. For instance, not all individuals can be evaluated simultaneously. The scalability of these PGAs is very good. If the population size increases, the total execution time can be kept constant by increasing the number of subpopulations. Moreover, a coarse-grained parallelization of GAs is the most straightforward

way to efficiently parallelize GAs. Two early examples of coarse-grained parallelism are described by Tanse (1987) and Pettey *et al* (1987).

In *fine-grained PGAs* the population is structured as in the isolation-by-distance model. The population is mapped onto a grid and a neighborhood structure is defined for each individual on this grid. The selection and the crossover step are restricted to the individuals in a neighborhood. The parallelism is obtained by assigning a parallel process to each individual. Communication between the processes is only necessary during selection and crossover.

Fine-grained PGAs are ideally suited for implementation on SIMD machines because each (identical) process operates on only one individual, and the communication between processes can be synchronized easily. Implementation on MIMD machines is also easy. Another advantage is that there is no need to introduce new parameters and insertion and deletion strategies to control migration. The diffusion of individuals over the grid implicitly controls the migration between demes.

Fine-grained PGAs offer a maximal amount of parallelism. Each individual can be evaluated and mutated in parallel. Moreover, because the neighborhood sizes are typically very small, the communication overhead is reduced to a minimum. The scalability is also maximal because additional individuals only imply more parallel processes which do not affect the total execution time. Early examples of this approach are described by Gorges-Schleuter (1989), Manderick and Spiessens (1989), Mühlenbein (1989), Hillis (1991), Collins and Jefferson (1991), and Spiessens and Manderick (1991).

E2.3.3 Dedicated hardware implementations for evolutionary algorithms

This section describes experimental hardware systems for GAs and classifier systems.

E2.3.3.1 Genetic algorithm hardware

Four experimental systems have been proposed or implemented so far: the *traveling salesman problem* (TSP) GA engine at the University of Tokyo, the WSI- (Wafer-Scale-Integration-) based GA machine at Tsukuba University, the GA processor at Victoria University, and the GA engine in EHW at the Electrotechnical Laboratory (ETL). G9.5

The *TSP machine* (Ohshima *et al* 1995) has been developed to see whether or not common algorithms such as GAs can be directly implemented in hardware. The GA for TSP has been successfully implemented using Xilinx's FPGAs (field programmable gate arrays). The order representation for city tours is used to avoid illegal genotypes.

The 16 GA engines are implemented on one board. Two GA engines are developed on one FPGA chip, a Xilinx 4010 chip with maximum 10 000 gates. The GA engine implements a transformation followed by the fitness evaluation part. The transformation translates the order representation of a city tour into the path representation of that tour. In the engine, the transformation, the GA operations, and the fitness evaluations are executed in a pipeline.

The improvement in execution time compared with a SPARC station2 (50 MHz) is expected to be about 800-fold. On a SPARC, 25 generations take 9.1 s, on the eight FPGAs (20 MHz) running in parallel they take only 4.7 ms.

The *WSI-based GA machine* (Yasunaga 1994) developed at Tsukuba University consists of 48 chromosome chips on a 5 in wafer having in total 192 chromosome processors. The basic idea is as follows. It is inevitable to have electrically defective areas in a large wafer. However, the robustness of the GA may absorb such defects since it is expected to work even if defective chromosome processors emerge.

The chromosome chips are connected in a two-dimensional array on the wafer. On a chromosome chip, four chromosome processors are also connected in an array structure.

At Victoria University in Australia, a GA processor was designed and partially implemented with Xilinx LCA chips (Salami 1995). Its applications include adaptive IIR filters and PID controllers. The GA processor architecture is described in the hardware description language VHDL. From this description, the logic circuits for the processor are synthesized.

EHW, described in section E2.3.4, is hardware which changes its own architecture using evolutionary computation. The EHW developed at the ETL is planned to include GA-dedicated hardware (Dirkx and Higuchi 1993) to cope with real-time applications. The key feature of this hardware is that GA operations

such as crossover and mutation are executed *bitwise* and in parallel at each chromosome (Higuchi *et al* 1994b). So far, PGAs have not attempted the parallelization of the GA operators themselves.

E2.3.3.2 Classifier system hardware

Although dedicated hardware systems for *classifier systems* have not yet appeared, the following four systems use associative memories or parallel machines in order to speed up the rule matching operations between the input message and the condition parts of the classifier rules. B1.5.2

Robertson's parallel classifier system, *CFS, was built on the Connection Machine (CM) which has an SIMD architecture with 64 000 1-bit processors. The speed of the system is independent of the number of classifiers (i.e. the execution speed is constant whenever the number of the rules is less than 65 000), but is dependent on the size of message list due to bit-serial processing algorithms of the CM (Robertson 1987).

Twardowski's learning classifier system is based on the associative memory architecture of Coherent Research Inc. The associative memory is used not only for the rule matching but also for other search operations such as parent selection (Twardowski 1994).

The GA-1 system is a parallel classifier system on the parallel associative processor IXM2 at ETL (Higuchi *et al* 1994a). IXM2, consisting of 73 transputers and a large associative memory (256 000 words), is used for rule matching. This is achieved one order of magnitude faster than on a Connection Machine-2 (Kitano *et al* 1991).

ALECSYS is a parallel software system, implemented on a network of transputers, that allows the development of learning agents with distributed control architecture (Dorigo 1995). An agent is modeled as a network of learning classifier systems (with the bucket brigade and a version of the GA) and is trained by reinforcement. ALECSYS has been applied to robot learning tasks in both simulated and real environments.

E2.3.4 Evolvable hardware

E2.3.4.1 Introduction

EHW is hardware which is built on FPGAs and whose architecture can be reconfigured by using evolutionary computing techniques to adapt to the new environment. If hardware errors occur or if new hardware functions are required, EHW can alter its own hardware structure in order to accommodate such changes.

Research on EHW was initiated independently in Japan and in Switzerland around 1992 (for recent overviews see Higuchi *et al* (1994b) and Marchal *et al* (1994), respectively). Since then, interest has been growing rapidly. For example, in Lausanne, October 1995, *EVOLVE95*, the first international workshop on EHW was held.

Research on EHW can be roughly classified into two groups: engineering oriented and embryology oriented. The engineering-oriented approach aims at developing a machine which can change its own hardware structure. It also tries to develop a new methodology for hardware design: hardware design without human designers. This group includes activities in ETL, ATR HIP Research Laboratories, and Sussex University.

The embryology-oriented approach aims at developing a machine which can selfreproduce or repair itself. Research in the Swiss Federal Institute of Technology and ATR is along this approach. Both are based on two-dimensional cellular automata.

After a brief introduction of FPGAs and the basic idea of EHW, the current research activities on EHW will be described.

E2.3.4.2 Field programmable gate arrays

An FPGA is a software-reconfigurable logic device whose hardware structure is determined by specifying a binary bitstring. The FPGA is the basis of EHW and it is described below.

The advantage of FPGAs is that only a short time is needed to realize a particular design or to change that design compared with ordinary gate arrays such as mask programmable gate arrays (MPGAs).

When some changes in the design are needed, the hardware description is revised using a textual hardware description language (HDL) and the new description is translated into a binary string. Then, that

string is downloaded into the FPGA and the new hardware structure is instantaneously built on the FPGA. Since they are so easy to reconfigure, FPGAs are becoming very popular especially for prototyping.

The structure of an FPGA is shown in figure E2.3.1. It consists of *logic block*s and *interconnection*s. Each logic block can implement an arbitrary hardware function depending on the specified bit string associated with that logic block. Another bitstring specifies which blocks can communicate over the interconnections. Thus, two bitstrings determine the hardware function of the FPGA and all these bits together are called the *architecture bit*s.

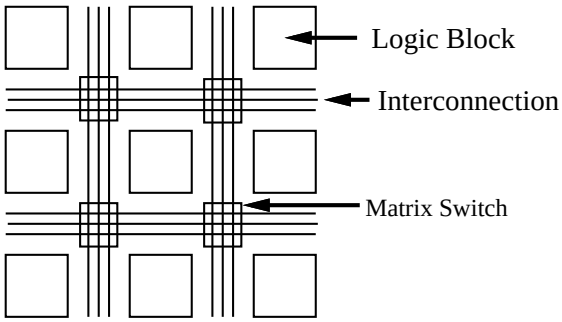


Figure E2.3.1. The architecture of an FPGA.

E2.3.4.3 The basic idea behind evolvable hardware

In EHW, genotype representations of hardware functions are finally transformed into hardware structures on the FPGA. There are various types of representation.

For example, the basic idea of ETL's EHW is to regard the architecture bits of the FPGAs as genotypes which are manipulated by the GA. The GA searches for the most appropriate architecture bits. Once a good genotype is obtained, it is then directly mapped on the FPGA (Higuchi *et al* 1994b), as shown in figure E2.3.2.

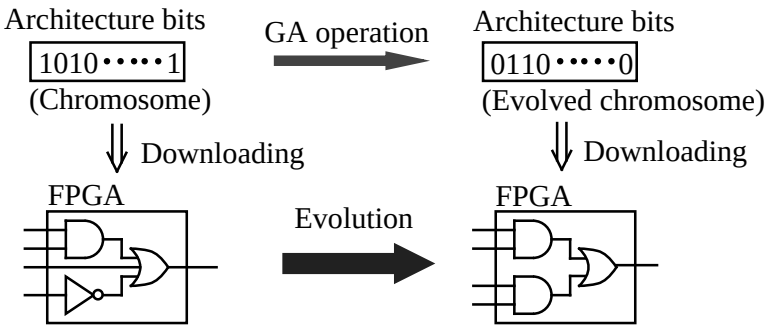


Figure E2.3.2. Hardware evolution by a GA.

The hardware evolution above is called *gate level evolution* because each gene may correspond to a primitive gate such as an AND gate.

E2.3.4.4 The engineering-oriented approach

Three research activities along the engineering-oriented approach are described here.

Since 1992, ETL has conducted research on gate level evolution and developed two application systems. One is the prototypical welding robot of which the control part can be taken over by EHW when a hardware error occurs. By the GA, EHW learns the target circuit without any knowledge about the circuit while it is functioning correctly. The other is a flexible pattern recognition system which shares the robustness of *artificial neural networks* (ANNs) (i.e. noise immunity). While neural networks learn D1 noise-insensitive functions by adjusting their weights and thresholds of neuron units, EHW implements

such functions directly in hardware by genetic learning. Recently ETL initiated *function level evolution* where each gene corresponds to real functions such as floating multiplication and sine functions. Function level evolution can attain performance comparable to that of neural networks (e.g. two-spirals) (Murakawa *et al* 1996). For this evolution, a dedicated ASIC (application specific integrated circuit) chip is being developed.

Also, Thompson at the University of Sussex evolves at gate level a *robot controller* using a GA. G3.7
For example, he evolved a 4 kHz oscillator and a finite-state machine that implements wall-avoidance behavior of a robot. The oscillator consisted of about 100 gates. The functions of the gates and their interconnections were determined by the GA (Thompson 1995).

Hemmi at ATR evolves the hardware description specified in the HDL by using genetic programming. The HDL used is SFL, which is a part of the LSI computer aided design (CAD) system, PARTHENON. Therefore, once such a description is obtained, the real hardware can be manufactured by PARTHENON (Hemmi *et al* 1994).

Hardware evolution proceeds as follows. The grammar rules of SFL are defined first. The order of application of the grammar rules specifies a hardware description. Then, a binary tree is generated which represents the order of rule application. Genetic programming uses these trees then to evolve better ones. So far, circuits such as adders have been successfully obtained.

E2.3.4.5 The embryology-oriented approach

Ongoing research at the Swiss Federal Institute of Technology aims at developing an FPGA (see section E2.3.4.2), which can selfreproduce or selfrepair (Marchal *et al* 1994).

A most interesting point is that the hardware description is represented by a binary decision diagram (BDD) and that these BDDs are treated as genomes by the GA. Each logical block of the FPGA reads the part of the genome which describes its function and is reconfigured accordingly. If some block is damaged, the genome can be used to perform selfrepair: one of the spare logical blocks will be reconfigured according to the description of the damaged block.

Other research according to the embryology-oriented approach is de Garis' work at ATR. His goal is to evolve neural networks using a two-dimensional *cellular automaton* machine (MIT CAM8 machine) G1.6
towards building an artificial brain. The neural network is formed as the trail on two-dimensional cellular automata by evolving the state transition rules by the GA (de Garis 1994).

E2.3.5 Conclusion

As EA computation has inherent parallelisms, EA computation using parallel machines is a versatile and effective way for speeding up. However, the developments of dedicated hardware systems to EA computation are limited to experimental systems and this situation would not change drastically. This is because the development of dedicated systems may restrict careful tuning of parameters and strategies for selection and recombination, which affects EA performance considerably. If killer applications for EA are found, dedicated hardware systems will be developed more in the future.

Another new research area, EHW, has a strong potential to explore new challenging applications which have not been handled well so far due to the requirements of adaptive change and real-time response. These applications may include multimedia communications such as asynchronous transfer mode (ATM) and adaptive digital processing/communications. To be used in practice in such areas, current EHW needs to find faster learning algorithms and killer applications.

References

Collins R J and Jefferson D R 1991 AntFarm: towards simulated evolution *Artificial Life II* ed C G Langton, C Taylor, J D Farmer and S Rasmussen (Redwood, CA: Addison-Wesley) pp 579–602

de Garis H 1994 An artificial brain—ATR's CAM-brain project aims to build/evolve an artificial brain with a million neural net modules inside a trillion cell cellular atutomata machine *New Generation Computing* vol 12 (Berlin: Springer) pp 215–21

Dirkx E and Higuchi T 1993 *Genetic Algorithm Machine Architecture* Matsumae International Foundation 1993 Fellowship Research Report, pp 225–36

Dorigo M 1995 ALECSYS and the autonoMouse: learning to control a real robot by distributed classifier systems *Machine Learning* vol 19 (Amsterdam: Kluwer) pp 209–40

Dorigo M and Sirtori E 1991 Alecsys: a parallel laboratory for learning classifier systems *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, July 1991)* ed R K Belew and L B Booker (San Mateo, CA: Morgan Kaufmann) pp 296–302

Fisher R A 1930 *The Genetical Theory of Natural Selection* (New York: Dover)

Gorges-Schleuter M 1989 ASPARAGOS: an asynchronous parallel genetic optimization strategy *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, June 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 422–7

Hemmi H, Mizoguchi J and Shimohara K 1994 Development and evolution of hardware behaviors *Proc. 4th Int. Workshop on the Synthesis and Simulation of Living Systems* ed R A Brooks and P Maes (Cambridge, MA: MIT Press) pp 371–6

Higuchi T, Handa K, Takahashi N, Furuya T, Iida H, Sumita E, Oi K and Kitano H 1994a The IXM2 parallel associative processor for AI *Computer* vol 27 (Los Alamitos, CA: IEEE Computer Society) pp 53–63

Higuchi T, Iba H and Manderick B 1994b Evolvable hardware *Massively Parallel Artificial Intelligence* ed H Kitano and J Hendler (Cambridge, MA: MIT Press) pp 398–421

Hillis W D 1991 Co-evolving parasites improve simulated evolution as an optimization procedure *Artificial Life II* ed C G Langton, C Taylor, J D Farmer and S Rasmussen (Redwood, CA: Addison-Wesley) pp 313–24

Kitano H, Smith S and Higuchi T 1991 GA-1: a parallel associative memory processor for rule learning with genetic algorithms *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, July 1991)* ed R K Belew and L B Booker (San Mateo, CA: Morgan Kaufmann) pp 296–302

Manderick B and Spiessens P 1989 Fine-grained parallel genetic algorithms *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, June 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 428–33

Marchal P, Piguet C, Mange D, Stauffer A and Durand S 1994 Embryological development on silicon *Proc. 4th Int. Workshop on the Synthesis and Simulation of Living Systems* ed R A Brooks and P Maes (Cambridge, MA: MIT Press) pp 365–70

Mühlenbein H 1989 Parallel genetic algorithms, population genetics and combinatorial optimization *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, June 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 416–21

Murakawa M, Yoshizawa S, Kajitani I, Furuya T, Iwata M and Higuchi T 1996 Hardware evolution at function levels *Proc. 4th Int. Conf. on Parallel Problem Solving from Nature (Berlin, 1996)* (*Lecture Notes in Computer Science 1141*) ed H-M Voigt, W Ebeling, I Rechenberg and H-P Schwefel (Berlin: Springer) pp 62–71

Ohhigashi H and Higuchi T 1995 *Hardware Implementation of Computation in Genetic Algorithms* Electrotechnical Laboratory Technical Report

Ohshima R, Matsumoto N and Hiraki K 1995 Research on the reconfigurable engine for genetic computation *Proc. 3rd Japan. FPGA/PLD Design Conf.* (Tokyo: CMP Japan) pp 541–8 (in Japanese)

Pettey C B, Leuze M R and Grefenstette J J 1987 A parallel genetic algorithm *Proc. 2nd Int. Conf. on Genetic Algorithms (Cambridge, MA, 1987)* ed J J Grefenstette (Hillsdale, NJ: Erlbaum) pp 155–61

Robertson G 1987 Parallel implementation of genetic algorithms in a classifier systems *Genetic Algorithms and Simulated Annealing* ed L Davis (London: Pitman) pp 129–40

Salami M 1995 Genetic algorithm processor for adaptive IIR filters *Proc. IEEE Int. Conf. on Evolutionary Computing* (CD-ROM) (Casual) pp 423–8

Spiessens P and Manderick B 1991 A massively parallel genetic algorithm: implementation and first analysis *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, July 1991)* ed R K Belew and L B Booker (San Mateo, CA: Morgan Kaufmann) pp 279–85

Tanese R 1987 Parallel genetic algorithms for a hypercube *Genetic Algorithms and their Applications: Proc. 2nd Int. Conf. on Genetic Algorithms (Cambridge, MA, 1987)* ed J J Grefenstette (Hillsdale, NJ: Erlbaum) pp 177–83

Thompson A 1995 Evolving electronic robot controllers that exploit hardware resources *Proc. 3rd Eur. Conf. on Artificial Life* (Berlin: Springer) pp 640–56

Twardowski K 1994 An associative architecture for genetic algorithm-based machine learning *Computer* vol 27 (Los Alamitos, CA: IEEE Computer Society) pp 27–38

Wright S 1931 Evolution in Mendelian populations *Genetics* vol 16

Yasunaga M 1994 Genetic algorithms implemented by wafer scale integration—wafer scale integration by LDA (leaving defects alone) approach *Trans. Inst. Electron. Information Commun. Eng.* **J77-D-I** 141–8 (Tokyo: Institute of Electronics, Information and Communication Engineers) (in Japanese)

Further reading

1. Gordon V S and Whitley D 1993 Serial and parallel genetic algorithms as function optimizers *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 155–62

2. Spiessens P 1993 *Fine-Grained Parallel Genetic Algorithms: Analysis and Applications* PhD Thesis, AI Laboratory, Free University of Brussels

These are good starting points for PGAs. Both give an overview of different PGAs and have extensive bibliographies.

3. Higuchi T, Iba H and Manderick B 1994b Evolvable hardware *Massively Parallel Artificial Intelligence* ed H Kitano and J Hendler (Cambridge, MA: MIT Press) pp 398–421

4. Marchal P, Piguet C, Mange D, Stauffer A and Durand S 1994 Embryological development on silicon *Proc. 4th Int. Workshop on the Synthesis and Simulation of Living Systems* ed R A Brooks and P Maes (Cambridge, MA: MIT Press) pp 365–70

These articles provide recent overviews of EHW.

F1.1 Introduction

Thomas Bäck

In this chapter, Evolutionary Computation Applications, we give an overview of some of the most prominent application domains to which evolutionary computation methods are successfully applied. The sections of this chapter are written by experts in these application domains, and the goal of each section is to provide an overview of the corresponding group of problems, of evolutionary computation approaches to problems from this domain, and of alternative approaches for solving problems from this domain.

For each application domain the most important evolutionary computation approaches are reviewed with respect to the representation of solutions, genetic operators, parameters of the algorithms, and implementation details.

As far as possible, a performance comparison between the evolutionary computation methods is performed, and the advantages and disadvantages of each of the different methods are summarized—especially regarding their performance in contrast to classical methods.

Typical groups of problems discussed in this chapter include control, identification, scheduling, pattern recognition, packing, simulation models, decision making, and simulated evolution. Some of these cover a broad range of problems (e.g. control), while others are more precisely defined by a number of properties that are characteristic for this application domain (e.g. simulation models). In any case, all problems from the same application domain share a number of properties that motivate an overview section on applying evolutionary computation in this domain.

Particular problems from an application domain may occur in various disciplines or branches of industry. Such case studies where a specific problem instance is discussed in full detail—including the design, development, implementation, practical aspects, and results of the evolutionary algorithm—are presented in Part G of this handbook. Of course, many of the problems presented there are just representatives of one of the problem domains discussed here, but representatives from the same domain may occur in different disciplines (i.e. chapters of Part G).

To summarize, the reader is encouraged to read the appropriate section in Part F if she is interested in an overview of a particular domain of problems. On the other hand, if a specific, detailed application example is required, the reader is advised to look for a representative case study in Part G of this handbook. To simplify this, table F1.1.1 provides an overview of those case studies that are instances of one of the application domain sections of Part F; the table is not meant to be comprehensive.

Table F1.1.1. Assignment of case studies in Part G to application domains in Part F.

	Application domain	Case studies
F1.2	Classical optimization problems	G9.2, G9.4, G9.5, G9.6, G9.7, G9.8, G9.10
F1.3	Control	G3.4
F1.4	Identification	G1.4, G1.6, G4.3
F1.5	Scheduling	G9.3, G9.4
F1.6	Pattern recognition	G8.1, G8.2
F1.7	The packing problem	
F1.8	Simulation models	
F1.9	Multicriterion decision making	
F1.10	Simulated evolution	

F1.2 Management applications and other classical optimization problems

Volker Nissen

Abstract

In this section an evaluation of the current situation regarding evolutionary algorithms (EAs) in management applications and classical optimization problems is attempted. References are divided into three categories: practical applications in management, application-oriented research in management, and standard optimization problems with relevance beyond the domain of management. Some general observations on the competitiveness of EAs, as compared to other optimization techniques, are also given. Few systematical and large-scale comparisons have appeared in the literature so far, and it is fair to state that a thorough evaluation of the potential of EAs in most of the classical optimization problems is still ahead of us. This is partly due to the lack of suitable benchmark problems, representative for distinct and neatly specified problem classes. Besides, theoretical results also shed a rather critical light on the objectives and current practice of empirical comparisons.

F1.2.1 Introduction

In recent years, new heuristic techniques, some of them inspired by nature, have emerged which have proven successful in solving very diverse hard optimization problems. Evolutionary algorithms (EAs), *tabu search* (TS), and *simulated annealing* (SA) are probably the best known classes of these modern heuristics. They share common characteristics. For instance, they tolerate deteriorations of the attained solution quality during the search process to overcome local suboptima in complex search spaces. D3.5

In this section†, EAs are viewed as stochastic heuristics, applicable to a large variety of complex optimization problems. They are based on the mechanisms of natural evolution, imitating the phenomena of heredity, variation, and selection on an abstract level. The mainstream types of EA are:

- *genetic algorithms* (GAs) B1.2
- *genetic programming* (GP) B1.5.1
- *evolution strategies* (ESs) B1.3
- *evolutionary programming* (EP). B1.4

Research in EAs is growing rapidly. This has been most visibly documented in a number of conference proceedings (Grefenstette 1985, 1987, Schaffer 1989, Belew and Booker 1991, Schwefel and Männer 1991, Männer and Manderick 1992, Fogel and Atmar 1992, 1993, Sebald and Fogel 1994, Davidor *et al* 1994, Pearson *et al* 1995, Eshelman 1995, McDonnell *et al* 1995). The field becomes increasingly diversified and complex.

In an attempt to structure one important area of applied EA research, this paper gives an overview of EA in management applications, also covering other classical optimization problems with relevance beyond the domain of management. More than 850 references to current as well as finished research projects and practical applications are classified in Appendix B. (The references in this text are collected in a separate reference list, located before the appendices.) Although much effort has been devoted to

† This section is an updated and extended version of Nissen (1993, 1995).

collecting and evaluating as many references as possible, the list cannot be complete. Furthermore, it must be assumed that many applications remain unpublished for reasons of confidentiality. Hence, the results reported in section F1.2.2 might be unintentionally biased. However, it is hoped that others will find the classification of applications and extensive reference list helpful in their own research. Moreover, some general observations on the competiveness of evolutionary approaches as compared to other paradigms are included in section F1.2.3.

F1.2.2 An overview of evolutionary algorithm applications in management science and other classical optimization problems

F1.2.2.1 Some technical remarks

This overview is mainly based on an evaluation of the literature and information posted to the relevant e-mail discussion lists *Genetic Algorithms Digest* (ga-list-request@aic.nrl.navy.mil), *Evolutionary Programming Digest* (ep-list-request@magenta.me.fau.edu), *Genetic Programming List* (genetic-programming-request@cs.stanford.edu), the EMSS list (dduane@gmu.edu) on evolutionary models in the social sciences, and two other specialized lists on timetabling (ttp-request@cs.nott.ac.uk) and scheduling (gasched-owner@acse.sheffield.ac.uk) with EAs. Additional information was gathered by private communication with fellow researchers, consultants, software developers, and users of EAs in business.

Sometimes it was rather difficult to decide, on the basis of the literature reviewed, whether papers actually discussed a practical application in business (section F1.2.2.2) or 'just' application-oriented research (section F.1.2.2.3). When only test problems were discussed without reference to a practical project then no immediate practical background was assumed. This also applies to projects using historical real data. Application-oriented research in management, and other classical optimization problems (section F1.2.2.4) are two evaluations that refer to projects not linked to practical applications in business. The section on other classical optimization problems concerns management as well as different (e.g. technical) domains. A well-known example for such a general standard problem with applicability in different domains is the *traveling salesman problem* (TSP). G9.5

Multiple publications on the same project count as one application, but all evaluated references are given in the tables of Appendix A and are listed in the extensive bibliography of Appendix B. The year of earliest presentation of an application, as given in the tables, generally refers to the earliest source found, which might be personal communication preceding a publication. In some cases, authors (Koza, Michalewicz) have included all previously published material in easily accessible books or long papers. Here, only the overall references are cited in the reference list.

For the majority of cited references the original papers were available for investigation. In some cases, however, secondary literature had to be used, because it was impossible or too difficult to obtain the original sources. Some additional references may be found in the bibliographies compiled by Alander (1996a, b, c, d) and available through the Internet (ftp://ftp.uwasa.fi.directory.cs/report94-1).

In this section, and particularly in the tables, a unified view on the field of EAs is taken. Even though the GA community is by far the largest, it is probably true that any of the EA mainstream types could be applied to any of the fields discussed here. Generally, a good optimization technique will account for the properties and biases of the problem investigated. The most reasonable solution representation, search operators and selection scheme will, therefore, depend on the problem. In this context, the entire field of EAs may be thought of as some form of toolbox. Whether the result of EA design for a particular problem on the basis of such a toolbox is called a GA, GP, EP or an ES is not really important, and might even be hard to decide. However, in the following overview sections the frequency of certain EA mainstream types will be mentioned for reasons of completeness.

F1.2.2.2 Practical applications in management

An overview of practical management applications is given in table F1.2.1. To date the quantity and diversity of applications is still moderate if one compares with the huge variety of optimization problems faced in management. Besides, many systems refered to in table F1.2.1 must be considered prototypes. Although the information is hard to extract from the given data, the number of running systems actually applied routinely in daily practice is likely to be rather small.

Combinatorial optimization with a focus on scheduling is most frequent. The majority of applications appear in an industrial setting with emphasis on production (figure F1.2.1). This is not surprising, since

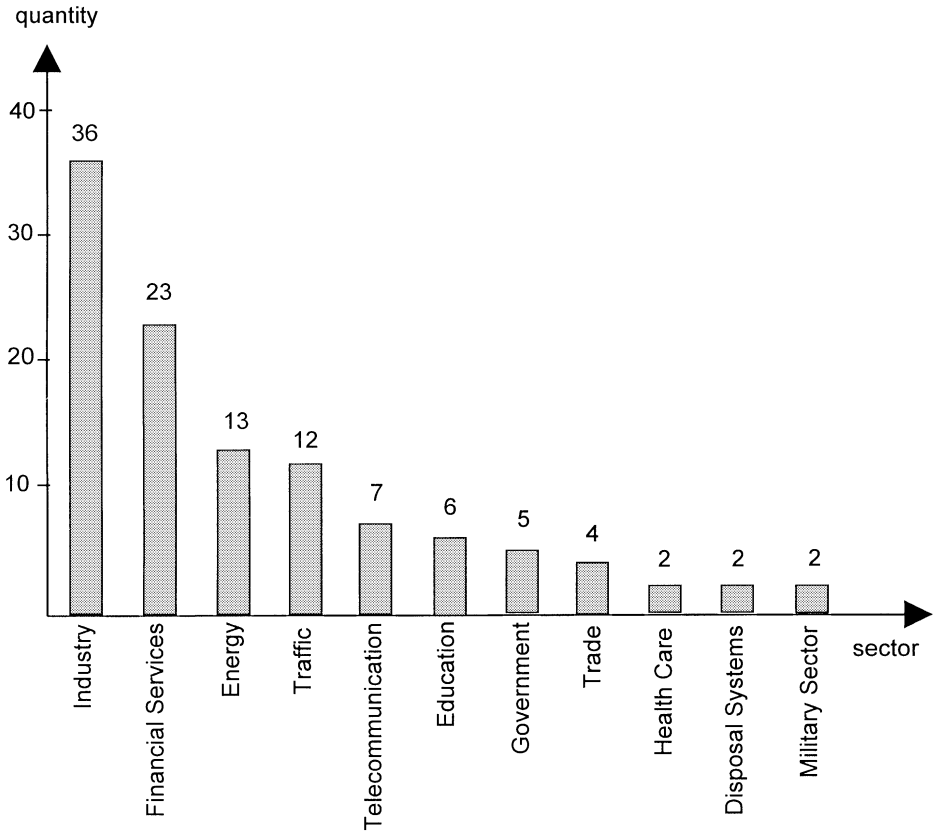


Figure F1.2.1. Practical applications ordered by economic sectors, as of July 1996.

production can be viewed as one large and complex optimization task that determines a company's competitive strength and success in business. Other business functions such as strategic planning, inventory, and marketing have not received much attention from the EA community so far, although some pioneering publications (see also table F1.2.2) have demonstrated the relevance of EAs to these fields.

The financial services sector is usually progressive in its electronic data processing applications, but publications in the scientific press are rather scarce. A focus on credit control and identification of good investment strategies is visible, though. The actual number of EA applications in this sector is likely to be *much* higher than the figures lead us to believe. This might also hold for management applications in the military sector. In these unpublished applications GAs are the most likely type of EA employed, since their research community is by far the largest.

The energy sector is another prevailing area of application. ESs are quite frequent here, because this class of EAs originated in the engineering field and has traditionally been strong in technical applications.

GAs are most frequently applied in practice. Interest in the other EA types is growing, however, so that a rise in the number of their respective applications can be expected in the near future. ESs and EP already cover a range of management-related applications. GP is a very recent technique that has attracted attention mainly from practitioners in the financial sector, while GP researchers are still working to reach the level of practical applicability in other domains.

Some hybrid systems integrating EAs with *artificial neural networks*, *fuzzy set theory*, and rule-based systems are documented. Since they are expensive to develop and may yield considerable strategic advantage over competitors, it can be assumed that much work in hybrid systems is kept secret and does not appear in the figures. This also holds for applications developed by commercial EA suppliers, sometimes with the aid of professional and semiprofessional EA tools. The quality of the data suffers from the fact that many authors are not allowed to publish their applications for reasons of confidentiality. D1, D2

If one considers the publication dates of practical EA applications (figure F1.2.2), a sharp rise in publications since the late 1980s is obvious. This movement can almost solely be attributed to an increased interest in GAs where the number of researchers has risen dramatically. To infer that GAs

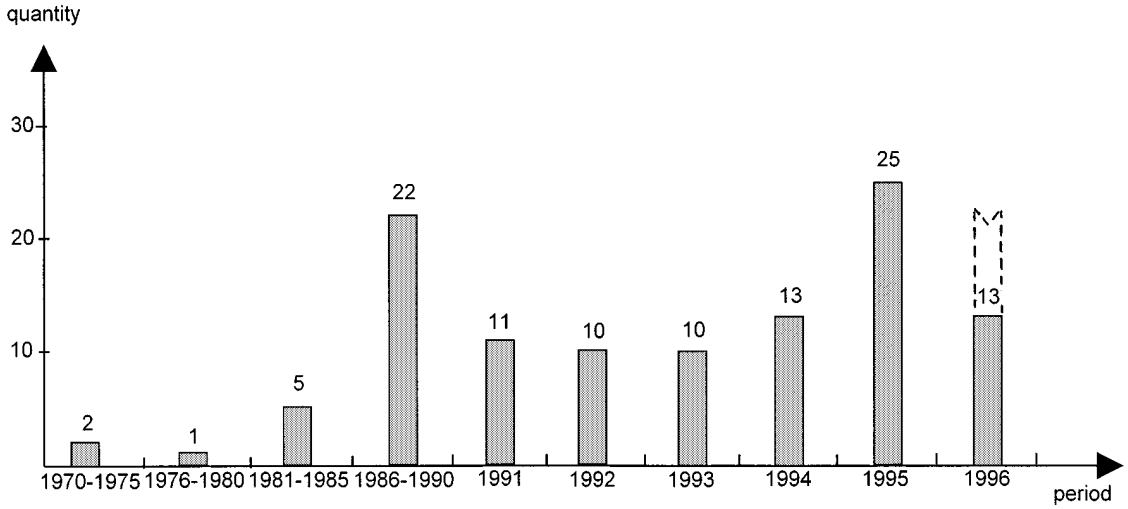


Figure F1.2.2. Practical applications ordered by earliest year of presentation as of July 1996.

are superior to other EA mainstream types can not be justified by these figures, though. It is rather the good 'infrastructure' of the GA community that fuels this trend: regular GA conferences since 1985, the availability of introductory textbooks, (semi-) professional GA tools, a well-organized and widely distributed newslist (*GA Digest*), and cumulative effects following successful pilot applications.

All in all, it seems fair to say that we have not seen the big breakthrough of EAs in management practice, yet. Interest in these new techniques, however, has risen considerably since 1990 and will lead to a further increase in practical applications in the near future.

F1.2.2.3 Application-oriented research in management science

This evaluation (table F1.2.2) focuses on research in management science that is not linked to any practical project in business. There is a strong focus on GAs, even more than in practical applications. The overall picture with respect to major fields of interest and EA types used is similar to that of the previous section. However, the quantity and diversity of projects is larger than in practical applications. Research interest in production planning and financial services is particularly high.

Notable is the strong bias of research for jobshop and flowshop scheduling. Production planning is an important problem in practice, of course. However, the standard test problems used by many authors frequently lack many of the practical constraints faced in production (see also section F1.2.3). Research on standard operations research problems such as jobshop scheduling sometimes seems to be some sort of tournament where the practical relevance of the approach comes second to minimal improvements of some published benchmark results on simplifying test problems.

F1.2.2.4 Other classical optimization problems

Table F1.2.3 lists EA applications on classical optimization problems with relevance to not only management science but other domains as well. Many of them refer to randomly generated data or benchmark problems given in the literature. The interested reader will find some applications from evolutionary economics under the heading *iterated games*.

Besides GAs (most frequent) and ESs, some applications of EP, GP and learning classifier systems are found in the area of game theory, as well as in some combinatorial problems such as the TSP. The TSP is a particularly well-studied problem that has led to the creation of a number of specialized recombination operators for GAs. The potential of GAs for the field of combinatorial optimization is generally considered to be high, but there has been some scientific dispute on this theme (see *GA Digest* **7** (1993), issue 6 and subsequent issues).

F1.2.3 Some general observations on the competitiveness of evolutionary algorithms

F1.2.3.1 Mixed results

Given the limited space available, it is impossible to discuss here in detail the implementations, advantages and disadvantages of EAs in particular optimization problems. However, some rather general observations will be presented that follow from the published literature, personal experience, and discussions of the author with developers and users of EAs.

Only a few systematic and large-scale empirical comparisons between EAs and other solution techniques appear in the literature. The most recent and quite extensive investigation was carried out by Baluja (1995). He compares seven iterative and evolution-based optimization techniques on 27 static optimization problems. The problem set includes jobshop scheduling, TSP, knapsack, binpacking, neural network weight optimization, and standard numerical function optimization tasks. Such problems are frequently investigated in the EA literature. Two GAs, three variants of multiple-restart stochastic iterated hillclimbing, and two versions of population-based incremental learning are compared in terms of speed and the best solution found in a given number of trials. The experiments indicate that algorithms simpler than standard GAs can perform competitively on both small and large problem instances.

Other empirical studies support these results. For instance, the investigations by Park and Carter (1995), Park (1995), Goffe *et al* (1994), Ingber and Rosen (1992), and Nissen (Section G9.10 of this handbook) all show no advantage or even disfavor EAs over SA and the related threshold accepting heuristic on classical optimization problems such as the Max-Clique, Max-Sat, and quadratic assignment problems. G9.10

In contrast, many examples can be found in the literature where evolutionary approaches compete successfully with the best solution techniques available so far. We only mention the works of Falkenauer on *binpacking* and grouping problems (Falkenauer and Delchambre 1992, Falkenauer 1994, 1995), Khuri *et al* on vertex cover and *multiple-knapsack* problems (Khuri *et al* 1994, Khuri and Bäck 1994), Lienig and Thulasiraman on routing tasks (Lienig and Thulasiraman 1994), Fleurent and Ferland on the *quadratic assignment problem* (Fleurent and Ferland 1994), and Parada Daza *et al* on the two-dimensional guillotine cutting problem (Parada Daza *et al* 1995). Moreover, the author knows of further practical applications of EAs in business where excellent results were produced in highly constrained complex search spaces. F1.7 G9.7 G9.10

These rather mixed results pose a problem for practitioners in search of the most promising optimization technique for a given hard problem. On the one hand, the current situation reflects the enormous difficulties associated with empirical crossparadigm comparisons. These difficulties concern benchmark problems and benchmark results. On the other hand, theoretical evidence suggests that the quest for a universally superior optimization technique is ill directed. The following sections take up these issues in some more depth.

F1.2.3.2 Benchmark problems

The first requirement for a systematic empirical comparison of different optimization methods is a representative set of instances for the investigated problem class. This in turn demands the neat specification and description of the relevant characteristics of this class. As Berens (1991) correctly points out, the success of an optimization method may change drastically when parameters of the given problem class are varied. Examples of such parameters are the problem size as well as structural aspects (such as symmetry and variance of entries in data matrices).

Moreover, real-world applications often involve multiple goals, noisy or time-varying objective functions, ill-structured data, and complex constraints that are usually not covered by standard test problems available today. Thus, if one does not want to be restricted to trivial toy problems many details can be necessary to correctly specify a problem class, and a sizeable number of problem instances might be required to cover the class representatively. As an example, Brandeau and Chiu (1989) have identified 33 characteristics to specify location problems. The complexity of creating meaningful benchmark problems is further raised by including aspects such as deception, epistasis, and related characteristics commonly used to establish the EA hardness of a problem.

At present, we are far from having suitable problem class descriptions and publicly available *representative* benchmark problems on a broad scale. The necessity to collect or generate them is generally acknowledged, though. Beasley's OR library of test problems (1990), available through the Internet from Imperial College in London, is a step in the right direction (http://mscmga.ms.ic.ac.uk/info.html). However,

it should be noted that it is extremely difficult to validate the suitability of any finite set of benchmark problems.

F1.2.3.3 Benchmark results

For a meaningful empirical comparison of competing optimization methods comparable statistical data are required. This is far from trivial. Several decisions must be taken in setting up the empirical test.

Choosing the right competitors. The comparison will have only limited significance unless we compare our approach with the strongest competitors available. It can require considerable effort to establish what paradigms should be compared. One reason is that certain very promising new heuristic techniques such as threshold accepting (Dueck and Scheuer 1990, Nissen and Paul 1995) are not widely known, yet. Others, such as tabu search and neural network approaches, have only been tested on a limited subset of classical optimization tasks, although they are potentially powerful in other problem classes as well.

Use results from the literature, or implement all compared paradigms? Implementing each optimization technique and performing experiments on the problem data is a very laborious task. Moreover, precise descriptions of every important detail of all compared paradigms are required. It is frequently difficult to obtain these precise descriptions from the literature. Even worse, as Koza points out in a recent posting to the *GP List*, one usually cannot avoid an unintentional bias in favor of the approach one is particularly familiar with.

However, suitable statistical data cannot in most cases be extracted from the literature. Authors use different measures to characterize algorithmic performance, such as the best solution found, mean performance, and variance of results. The number of runs to obtain statistical data for a given optimization method can vary between 1 and 100 in the literature. Moreover, differing hardware and software makes efficiency comparisons between own data and published results difficult.

Asking authors for the code that was used in generating published benchmark results can also lead to many difficulties related to program documentation, programming style, or hardware and software requirements.

Algorithmic design and parameter settings. There are numerous published variants of EAs, particularly concerning GAs. GAs were originally not developed for function optimization (De Jong 1993). However, much effort has been devoted to adapting them to optimization tasks, especially in terms of representation and search operators. Additional algorithmic parameters such as population size and population structure, crossover rate, and selection mechanism result in a considerable design flexibility for the developer. The same applies, albeit to a lesser extent, to other optimization methods one wishes to investigate.

This freedom in designing the optimization techniques and the difficulty of determining adequate strategy parameter settings adds further complexity to crossparadigm comparisons. It is impossible to test every design option. Additionally, there are different opinions as to whether a fair empirical comparison should focus on the generality of a method over many problem classes, or the power in one specific area of application. Generally, a tradeoff between the power and the generality of a solution technique will be observed (Newell 1969). Baluja (1995), for instance, who disfavors GAs, concentrates on generality. Successful evolutionary approaches, on the other hand, frequently apply a highly problem-dependent representation or decoding scheme and search operators, or they use hybrid approaches that combine EAs with other techniques (see, for example, the works of Davis (1991), Mühlenbein (1989), Liepins and Potter (1991), Falkenauer (1995), and Fleurent and Ferland (1994, 1995)). This leads to the next difficult decision.

Quality indicators for comparisons of optimization techniques. Besides the characteristics of power and generality there are many other aspects of an optimization technique that could be used to assess its quality. Examples include efficiency and ease of implementation. Matters are further complicated in that even the definition and measurement of these quality indicators is not universally agreed upon.

Conduct of the empirical comparison. The general setup of the experiments is crucial for the validity of results. Important decisions include the method of initialization, the termination criterion, and the number of runs on each problem instance.

Besides these difficulties in conducting meaningful empirical comparisons, theoretical results also suggest that it is hard to come to *general* conclusions about advantages and disadvantages of evolutionary optimization.

F1.2.3.4 Implications of the no-free-lunch theorem

Recently, Wolpert and Macready (1995) published a theorem that basically states the following (the no-free-lunch, NFL, theorem): *all algorithms that search for an extremum of a cost function perform exactly the same, when averaged over all possible cost functions.* This result is not specific to EAs but also concerns competing optimization methods.

Some very practical consequences follow from this theorem. They are not really new to optimization practitioners. However, the NFL theorem provides some useful theoretical background.

- The quest for an optimization technique that is generally superior is ill directed, as long as the area of application is not narrowly and precisely defined.
- Good performance of an optimization technique in one area of application will not guarantee equally good results in a different problem area.
- It is necessary to account for the particular biases and properties of the given cost function in the design of a successful algorithm for this application. In other words, one should start by analyzing the problem before thinking about the proper solution technique. Empirical comparisons, however, frequently proceed in the opposite direction, taking some broadly applicable optimization techniques and then looking for suitable test problems.

It is hard to come to general conclusions on advantages and disadvantages of EAs, given the NFL theorem and the difficult empirical situation. The statements in the following section should be taken as the author's subjective view.

F1.2.3.5 Some advantages and disadvantages of evolutionary algorithms

To start with an advantage, it is not difficult to explain the basic idea of EAs to somebody completely new to the field. This is of great importance in terms of practical acceptance of the evolutionary approach.

An advantage and disadvantage at the same time is the design flexibility of EAs. It allows for adaptation to the problem under study, and the breadth of known EA applications gives testimony to this. EAs have in a relatively short time demonstrated their usefulness on an impressive variety of difficult optimization problems, including time-varying and stochastic environments. Algorithmic design of an EA can be achieved in a stepwise, prototyping-like manner. It is easy to produce a first working implementation that can then be improved upon, including domain-specific knowledge and using the 'EA toolbox' mentioned before. This adaptation of the method, however, requires empirical testing of design options and a sound methodical knowledge. In this sense, the many strategy parameters of today's EAs are clearly a disadvantage, as compared to simpler competing optimization methods.

The basic EA types are broadly applicable and, in contrast to many of the more traditional optimization techniques, make only weak assumptions about the domain under study. They can be applied even when the insight into the problem domain is low. In fact, EAs can be positioned along a continuum from weak, broadly applicable methods to strong, highly specialized methods. (Compare also Michalewicz's hierachy of evolution programs (Michalewicz 1996).) Moreover, there are a variety of ways of integrating and hybridizing EAs with other existing methods, as evidenced by numerous publications. These advantages will, however, in general also hold for similar modern heuristics, such as SA or tabu search, even though they might currently lag behind in terms of total research effort spent.

With these competitors EAs also share some disadvantages. First, EAs can generally offer no guarantee of identifying a global optimum in limited time. They are of heuristic character. However, in practical applications it is often not necessary to find a global optimum, but a good solution will suffice. Unfortunately, it is difficult to predict the solution quality attainable with EAs on arbitrary real-world problems in a given amount of time. More generally, the empirical success of EAs is not easily explained in terms of the EA theory available today.

The population approach of EAs usually leads to high computational demands. Since EAs are easily parallelized, this is becoming less of a problem as available hardware power increases and parallel computers are more and more common. Furthermore, the optimization process can be rather inefficient in

the final search phase, particularly for GAs. Hybridizing with a quick local optimizer can cater for this problem, though.

With a few exceptions (such as grouping problems), it seems very difficult today to predict in advance whether for a particular real-world optimization problem EAs will produce results superior to those of similar modern heuristics such as threshold accepting or SA. The most important point is really to account for the properties of the problem in designing the algorithm, and here EAs offer a large toolbox to choose from.

F1.2.4 Conclusions

Over the last couple of years, interest in EAs has risen considerably amongst researchers and practitioners in the management domain, although we have not seen the major breakthrough of EAs in practical applications, yet. Most people have been attracted by GAs, while ESs, EP, and GP are not so widely known. GP is the newest technique and is just reaching the level of practical applicability, particularly in the financial sector. Even though GAs are most common, this should not be interpreted as superiority over other EA types. It rather seems to be a good 'infrastructure' that contributes to the trend for GAs.

The majority of applications analyzed here concern GAs in combinatorial optimization. Many researchers focus on standard problems to test the quality of their algorithms. The results are mixed. This is partly due to the enormous difficulties associated with conducting meaningful empirical comparisons between optimization techniques. Moreover, the NFL theorem tells us that one should not expect to find a universally superior optimization method. However, the current efforts to develop professional EA tools and parallelize EA applications, and the exponentially growing number of EA researchers, will lead to more practical applications in the future and a better understanding of the relative advantages and weaknesses of the evolutionary approach. Figure F1.2.3 is an attempt to assess the current position of EAs as an optimization method with respect to the technological life cycle.

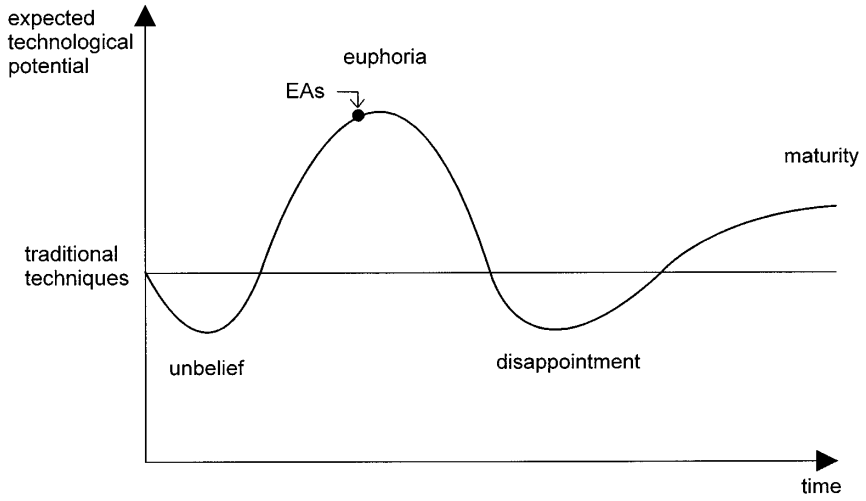


Figure F1.2.3. An estimation of the current state of EAs as an optimization method in a life cycle model as of July 1996.

There is evidence for the robustness of EAs in stochastic optimization where the evaluation involves noise or requires an approximation of the true objective function value (Grefenstette and Fitzpatrick 1985, Hammel and Bäck 1994, Nissen and Biethahn 1995). Encouraging first results have also been achieved in time-varying environments employing nonstandard concepts such as diploidy (Smith 1987, Smith and Goldberg 1992, Dasgupta and McGregor 1992, Ng and Wong 1995). EAs also have been shown to work well on integer programming problems which are presently difficult to solve with conventional techniques such as linear programming for large or nonlinear instances (Bean and Hadj-Alouane 1992, Hadj-Alouane and Bean 1992, Khuri *et al* 1994, Khuri and Bäck 1994, Rudolph 1994).

Currently, EAs are becoming more and more integrated as an optimization module in large software products (e.g. for production planning). Thereby, the end user is often unaware that an evolutionary approach to problem solving is employed. Integrating and hybridizing EAs with other techniques is a most promising research direction. It aims at combining the relative advantages of different problem solving methods and leads to powerful tools for practical applications.

References cited in the text

Alander J 1996a *An Indexed Bibliography of Genetic Algorithms in Operations Research* University of Vaasa Department of Information Technology and Production Economics Report Series 94-1-OR

——1996b *An Indexed Bibliography of Genetic Algorithms in Manufacturing* University of Vaasa Department of Information Technology and Production Economics Report Series 94-1-MANU

——1996c *An Indexed Bibliography of Genetic Algorithms in Logistics* University of Vaasa Department of Information Technology and Production Economics Report Series 94-1-LOGISTICS

——1996d *An Indexed Bibliography of Genetic Algorithms in Economics and Finance* University of Vaasa Department of Information Technology and Production Economics Report Series 94-1-ECO

Baluja S 1995 *An Empirical Comparison of Seven Iterative and Evolutionary Function Optimization Heuristics* Carnegie Mellon University School of Computer Science Technical Report CMU-CS-95-193

Bean J C and Hadj-Alouane A B 1992 *A Dual Genetic Algorithm for Bounded Integer Programs* University of Michigan College of Engineering, Department of Industrial and Operations Research Technical Report 92-53

Beasley J E 1990 OR-library: distributing test problems by electronic mail *J. Operational Res. Soc.* **41** 1069–72

Belew R K and Booker L B (eds) 1991 *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, July 1991)* (San Mateo, CA: Morgan Kaufmann)

Berens W 1991 *Beurteilung von Heuristiken* (Wiesbaden: Gabler)

Brandeau M L and Chiu S S 1989 An overview of representative problems in location research *Management Sci.* **35** 645–74

Dasgupta D and McGregor D R 1992 Nonstationary function optimization using the structured genetic algorithm *Proc. 2nd Conf. on Parallel Problem Solving from Nature (Brussels, 1992)* ed R Männer and B Manderick (Amsterdam: Elsevier–North-Holland) pp 145–54

Davidor Y, Schwefel H-P and Männer R (eds) 1994 *Parallel Problem solving from Nature—PPSN III (Proc. Int. Conf. on Evolutionary Computation and 3rd Conf. on Parallel Problem Solving from Nature, Jerusalem, October 1994)* (*Lecture Notes in Computer Science 866*) ed Yu Davidor, H-P Schwefel and R Männer (Berlin: Springer)

Davis L (ed) 1991 *Handbook of Genetic Algorithms* (New York: Van Nostrand Reinhold)

De Jong K A 1993 Genetic algorithms are NOT function optimizers *Foundations of Genetic Algorithms* vol 2, ed D Whitley (San Francisco, CA: Morgan Kaufmann) pp 5–17

Dueck G and Scheuer T 1990 Threshold accepting: a general purpose optimization algorithm appearing superior to simulated annealing *J. Comput. Phys.* **90** 161–75

Eshelman L J (ed) 1995 *Proc. 6th Int. Conf. on Genetic Algorithms (Pittsburgh, PA, July 1995)* (San Mateo, CA: Morgan Kaufmann)

Falkenauer E 1994 A new representation and operators for genetic algorithms applied to grouping problems *Evolutionary Comput.* **2** 123–44

——1995 Tapping the full power of genetic algorithm through suitable representation and local optimization: application to bin packing *Evolutionary Algorithms in Management Applications* ed J Biethahn and V Nissen (Berlin: Springer) pp 167–82

Falkenauer E and Delchambre A 1992 A genetic algorithm for bin packing and line balancing *Proc. 1992 IEEE Int. Conf. on Robotics and Automation (Nice, May 1992)* (Piscataway, NJ: IEEE) pp 1186–92

Fleurent C and Ferland J A 1994 Genetic hybrids for the quadratic assigment problem *DIMACS Series in Discrete Mathematics and Theoretical Computer Science* vol 16, ed P M Pardalos and H Wolkowicz (Providence, RI: American Mathematical Society) pp 173–88

——1995 Genetic and hybrid algorithms for graph coloring *Ann. Operations Res. Special Issue on Metaheuristics in Combinatorial Optimization* ed G Laporte, I H Osman and P L Hammer, at press

Fogel D B and Atmar W (eds) 1992 *Proc. 1st Ann. Conf. on Evolutionary Programming* ed D B Fogel and W Atmar (La Jolla, CA: Evolutionary Programming Society)

——1993 *Proc. 2nd Ann. Conf. on Evolutionary Programming (San Diego, CA, 1993)* ed D B Fogel and W Atmar (San Diego, CA: Evolutionary Programming Society)

Goffe W L, Ferrier G D and Rogers J 1994 Global optimization of statistical functions with simulated annealing *J. Econometrics* **60** 65–99

Grefenstette J J (ed) 1985 *Proc. Int. Conf. on Genetic Algorithms and their Applications (Pittsburgh, PA, 1985)* (Hillsdale, NJ: Erlbaum)

——1987 *Proc. 2nd Int. Conf. on Genetic Algorithms (Cambridge, MA, 1987)* ed J J Grefenstette (Hillsdale, NJ: Erlbaum)

Grefenstette J J and Fitzpatrick J M 1985 Genetic search with approximate function evaluations *Proc. Int. Conf. on Genetic Algorithms and their Applications (Pittsburgh, PA, 1985)* ed J J Grefenstette (San Mateo, CA: Morgan Kaufmann) pp 112–20

Hadj-Alouane A-B and Bean J C 1992 *A Genetic Algorithm for the Multiple-Choice Integer Program* University of Michigan College of Engineering Department of Industrial and Operations Research Technical Report 92-50

Hammel U and Bäck T 1994 Evolution strategies on noisy functions—how to improve convergence properties *Parallel Problem solving from Nature—PPSN III (Proc. Int. Conf. on Evolutionary Computation and 3rd Conf. on Parallel Problem Solving from Nature, Jerusalem, October 1994)* (*Lecture Notes in Computer Science 866*) ed Yu Davidor, H-P Schwefel and R Männer (Berlin: Springer) pp 159–68

Ingber L and Rosen B 1992 Genetic algorithms and very fast simulated annealing—a comparison *Math. Comput. Model.* **16** 87–100

Khuri S and Bäck T 1994 An evolutionary heuristic for the minimum vertex cover problem *Genetic Algorithms within the Framework of Evolutionary Computation. Proc. KI-94 Workshop* Max-Planck-Institut für Informatik Technical Report MPI-I-94-241, ed J Hopf, pp 86–90

Khuri S, Bäck T and Heitkötter J 1994 The zero/one multiple knapsack problem and genetic algorithms *Proc. 1994 ACM Symp. on Applied Computing* ed E Deaton, D Oppenheim, J Urban and H Berghel (New York: ACM) pp 188–93

Lienig J and Thulasiraman K 1994 A genetic algorithm for channel routing in VLSI circuits *Evolutionary Comput.* **1** 293–311

Liepins G E and Potter W D 1991 A genetic algorithm approach to multiple-fault diagnosis *Handbook of Genetic Algorithms* ed L Davis (New York: Van Nostrand Reinhold) pp 237–50

Männer R and Manderick B (eds) 1992 *Parallel Problem Solving from Nature, 2 (Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature, Brussels, 1992)* ed R Männer and B Manderick (Amsterdam: Elsevier)

McDonnell J R, Reynolds R G and Fogel D B (eds) 1995 *Proc. 4th Ann. Conf. on Evolutionary Programming (San Diego, CA, March 1995)* ed J R McDonnell, R G Reynolds and D B Fogel (Cambridge, MA: MIT Press)

Michalewicz Z 1996 *Genetic Algorithms + Data Structures = Evolution Programs* 3rd edn (Berlin: Springer)

Mühlenbein H 1989 Parallel genetic algorithms, population genetics and combinatorial optimization *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 416–21

Newell A 1969 Heuristic programming: ill structured problems *Progress in Operations Research—Relationships between Operations Research and the Computer* vol 3, ed J Aronofsky (New York: Wiley) pp 361–414

Ng K P and Wong K C 1995 A new diploid scheme and dominance change mechanism for non-stationary function optimization *Proc. 6th Int. Conf. on Genetic Algorithms (Pittsburgh, PA, July 1995)* ed L J Eshelman (San Mateo, CA: Morgan Kaufmann) pp 159–66

Nissen V 1993 *Evolutionary Algorithms in Management Science. An Overview and List of References* European Study Group for Evolutionary Economics Papers on Economics and Evolution 9303

——1995 An overview of evolutionary algorithms in management applications *Evolutionary Algorithms in Management Applications* ed J Biethahn and V Nissen (Berlin: Springer) pp 44–97

Nissen V and Biethahn J 1995 Determining a good inventory policy with a genetic algorithm *Evolutionary Algorithms in Management Applications* ed J Biethahn and V Nissen (Berlin: Springer) pp 240–50

Nissen V and Paul H 1995 A modification of threshold accepting and its application to the quadratic assignment problem *OR Spektrum* **17** 205–10

Parada Daza V, Muñoz R and Gómes de Alvarenga A 1995 A hybrid genetic algorithm for the two-dimensional guillotine cutting problem *Evolutionary Algorithms in Management Applications* ed J Biethahn and V Nissen (Berlin: Springer) pp 183–96

Park K 1995 A comparative study of genetic search *Proc. 6th Int. Conf. on Genetic Algorithms (Pittsburgh, PA, July 1995)* ed L J Eshelman (San Mateo, CA: Morgan Kaufmann) pp 512–19

Park K and Carter B 1995 On the effectiveness of genetic search in combinatorial optimization *Proc. 10th ACM Symp. on Applied Computing* GA & Optimization Track: 329–36

Pearson D W, Steele N C and Albrecht R F (eds) 1995 *Proc. Int. Conf. on Artificial Neural Nets and Genetic Algorithms (Alès, 1995)* (Berlin: Springer)

Rudolph G 1994 An evolutionary algorithm for integer programming *Parallel Problem solving from Nature—PPSN III (Proc. Int. Conf. on Evolutionary Computation and 3rd Conf. on Parallel Problem Solving from Nature, Jerusalem, October 1994)* (*Lecture Notes in Computer Science 866*) ed Yu Davidor, H-P Schwefel and R Männer (Berlin: Springer) pp 139–48

Schaffer J D (ed) 1989 *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, June 1989)* (San Mateo, CA: Morgan Kaufmann)

Schwefel H-P and Männer R (eds) 1991 *Proc. 1st Workshop on Parallel Problem Solving from Nature (PPSN I) (Dortmund, 1991)* (Berlin: Springer)

Sebald A V and Fogel L J (eds) 1994 *Proc. 3rd Ann. Conf. on Evolutionary Programming (San Diego, CA, February 1994)* ed A V Sebald (Singapore: World Scientific)

Smith R E 1987 Diploid genetic algorithms for search in time varying environments *Proc. Ann. Southeast Regional Conf. ACM* pp 175–9

Smith R E and Goldberg D E 1992 Diploidy and dominance in artificial genetic search *Complex Syst.* **6** 251–85

Wolpert D H and Macready W G 1995 *No Free Lunch Theorems for Search* Santa Fe Institute Technical Report SFI-TR-95–02–010

Appendix A. Tables

Tables F1.2.1, F1.2.2, and F1.2.3 list, respectively, the use of EAs in practical management applications, in application-oriented research in management science, and in other classical optimization problems. The references cited in these tables are listed in Appendix B. In tables F1.2.1 and F1.2.2, the 'Earliest known' column indicates the year of the earliest known presentation.

Table F1.2.1. Practical applications of EAs in management.

Economic sector	Practical application in business	References	Earliest known
1. Industry	Line balancing in the metal industry	[FALK92]	1992
1.1 Production		[FULK93b]	1993
	Simultaneous planning of production program,lotsizes, and production sequence in the wallpaper industry	[ZIMM85]	1984
	Load balancing for sugar beet presses	[FOGA95d], [VAVA95]	1995
	Balancing combustion between multiple burners in furnaces and boiler plants	[FOGA88, 89]	1988
	Grouping orders into lots in a foundry	[FALK91b]	1991
	Multiobjective production planning	[BUSC91] [NOCH90]	1991 1990
	Deciding on buffer capacity and number of system pallets in chained production	[NOCH90]	1990
	Production planning in the chemical industry	[BRUN93a]	1993
	Lotsizing and sequencing in the car industry	[ABLA91, 95a]	1989
	Flowshop scheduling for the production of integrated circuits	[WHIT91], [STAR92]	1991
	Sector release scheduling at a computer board assembly and test facility	[CLEV89]	1989
	Sequencing orders in the electrical industry	[ABLA90]	1989
	Sequencing orders in the paper industry	[ABLA90]	1989
	Scheduling foundry core–pour–mold operations	[FULK93a]	1993
	Sequencing in a hot-rolling process	[SCHU93b, c]	1992
	Sequencing orders for the production of engines	[SCHÖ90, 91, 92, 94]	1990
	Scheduler for a finishing plant in clothing	[FOGE96]	1996
	Process planning for part of a multispindle machine	[VÁNC91]	1991
	Stacking of aluminium plates onto pallets	[PROS88]	1988
	Production planning with dominant setup costs	[SCHÜ94]	1994
	Scheduling in car production of Daimler–Benz	[FOGA95a, b], [KRES96]	1995
	Scheduling (assumed: production scheduling) at Rolls Royce (application not confirmed by Rolls Royce)	[FOGA95a, b]	1995
	Sequencing and lotsizing in the pharmaceutical industry	[SCHU94]	1994
	Forge scheduling	[SIRA95]	1995

Table F1.2.1. Practical applications of EAs in management (continued).

Economic sector	Practical application in business	References	Earliest known
1.1 Production (continued)	Production scheduling in a steel mill	[YONG95]	1995
	Slab design (kind of bin packing)	[HIRA95]	1995
	Scheduling and resource management in ship repair	[FILI94, 95]	1994
	Just-in-time scheduling of a collator machine	[RIXE95], [KOPF95]	1995
	Optimizing the cutting of fabric	[FOGE96]	1996
1.2 Inventory	Inventory control in engine manufacturing	[FOGE96]	1996
1.3 Personnel	Crew/staff scheduling	[WILL96]	1996
	Crew scheduling in an industrial plant	[MOCC95]	1995
1.4 Distribution	Siting of retail outlets	[HUGH90]	1990
	Allocation of orders to loading docks in a brewery	[STAR91b, 92, 93a, b]	1991
2. Financial services	Assessing insurance risks	[HUGH90]	
	Developing rules for dealing in currency markets	[HUGH90]	1990
	Modeling trading behavior in financial markets	[SCHU93a]	1993
	Trading strategy search	[NOBL90]	1990
	Security selection and portfolio optimization	[NOBL90]	1990
	Risk management	[NOBL90]	1990
	Evolved neural network predictor to handle pension money	[FOGE96]	1996
	Credit scoring	[NOBL90], [WALK94, 95]	1990 1994
	Time series analysis	[NOBL90]	1990
	Credit card application scoring	[FOGA91, 92]	1991
	Credit card account performance scoring	[FOGA91, 92]	1991
	Credit card transaction fraud detection	[FOGA91, 92]	1991
	Credit evaluation at the Co-Operative Bank	[KING95]	1995
	Fraud detection at Travelers Insurance Company	[KING95]	1995
	Fraud detection at the Bank of America	[VERE95b]	1995
	Financial trading rule generation	[KERS94]	1994
	Detecting insider dealing at London Stock Exchange	[KING95]	1995
	Building financial trading models	[ROGN94]	1994
	Improving trading strategies in stock market simulation	[MAZA95]	1995
	Prediction of prepayment rates for adjustable-rate home mortgage loans	[VERE95a]	1995
	Optimal allocation of personnel in a large bank	[EIBE94b]	1994
	Constructing scorecards for credit control	[FOGA94b], [IRES94, 95]	1994

Table F1.2.1. Practical applications of EAs in management (continued).

Economic sector	Practical application in business	References	Earliest known
3. Energy	Optimal load management in a power plant network	[ADER85], [WAGN85] [HOEH96]	1985 1996
	Optimized power flow in energy supply networks	[MÜLL83a, b, 86] [FUCH83]	1983
	Cost-efficient core design of fast breeder reactors	[HEUS70]	1970
	Optimizing a chain of hydroelectric power plants	[HÜLS94]	1994
	Scheduling planned maintenance of the UK electricity transmission network	[LANG95, 96]	1995
	Network pipe sizing for British Gas	[SURR94, 95]	1994
	Refueling of pressurized water reactors	[AXMA94a, b], [BÄCK95, 96a]	1994
	Scheduling in a liquid-petroleum pipeline	[SCHA94]	1994
	Hot parts operating scheduling of gas turbines	[SAKA93]	1993
	Maximizing efficiency in power station cycles	[SONN82]	1981
	Scheduling delivery trucks for an oil company	[FOGE96]	1996
4. Traffic	Routing and scheduling of freight trains	[GABB91]	1991
	Scheduling trains on single-track lines	[MILL93], [ABRA93b, 94]	1993
	Vehicle routing (United Parcel Service)	[KADA90a, b, 91]	1990
	Finding a just-in-time delivery schedule	[KEME95b]	1995
	Multicommodity transshipment problem	[THAN92b]	1992
	School bus routing	[THAN92a]	1992
	Determining railtrack reconstruction sites to minimize traffic disturbance	[ABLA92, 95a]	1992
	Scheduling cleaning personnel for trains	[ABLA92, 95a]	1992
	Scheduling aircraft landing times to minimize cost	[ABRA93a]	1993
	Predicting the bids of pilots for promotion to larger than their current aircraft	[SYLO93]	1993
	Elevator dispatching	[SIRA95]	1995
	Vehicle scheduling problem of the mass transportation company of Mestre, Italy	[BAIT95]	1995
5. Telecommunication	Designing low-cost sets of packet switching communication network links	[DAVI87, 89], [COOM87]	1987
	Anticipatory routing and scheduling of call requests	[COX91]	1991
	Designing a cost-efficient telecommunication network with a guaranteed level of survivability	[DAVI93b]	1993
	Local and wide-area network design	[KEAR95]	1995
	TSP for several system installers	[KEAR95]	1995
	Optimizing telecommunication network layout	[KEIJ95]	1995
	On-line reassignment of computer tasks across a suite of heterogeneous computers	[FOGE96]	1996

Table F1.2.1. Practical applications of EAs in management (continued).

Economic sector	Practical application in business	References	Earliest known
6. Education	School timetable problem	[COLO91a, b, 92a] [LING92b]	1990 1992
	Scheduling student presentations	[PAEC94b]	1994
	Hybrid solution for a polytechnic timetable problem	[LING92a]	1992
	Timetabling of exams and classes	[ERGU95]	1995
	Exam scheduling problem	[CORN93, 94], [ROSS94a, b]	1993
7. Government	Optimizing budgeting rules by data analysis	[PACK90]	1990
	Automatically screening tax claims	[KING95]	1995
	Scheduling the Hubble Space Telescope	[SPON89]	1989
	Mission planning (two cases)	[FOGE96]	1996
8. Trade	Determining cluster storage properties for product clusters in a distribution center for vegetables/fruits	[BROE95a, b]	1995
	Data mining—analyzing supermarket customer data	[KOK96]	1996
	Optimal selection for direct mailing in marketing	[EIBE96]	1996
	Determining the right quantity of books' first editions	[ABLA95a]	1995
9. Health care	Scheduling patients in a hospital	[ABLA92, 95a]	1992
	Allocating investments to health service programs	[SCHW72]	1972
10. Disposal systems	Optimal siting of local waste disposal systems	[FALK80]	1980
	Vehicle routing and location planning for waste disposal systems	[DEPP92]	1992
11. Military sector	Mission planning	[FOGE96]	1996
	Scheduling an F-14 flight simulator to pilots	[SYSW91a, b]	1991

Table F1.2.2. EAs in application-oriented research in management science. The third column, headed 'No', indicates the number of projects.

General topic	Research application	No	References	Earliest known
Location problems	Facility layout/location planning	10	[KHUR90], [TAM92], [SMIT93], [YIP93], [CHAN94a], [CONW94], [NISS94a, b], [YERA94], [KADO95], [KRAU95], [GARC96]	1990–1996
	Layout design	1	[STAW95]	1995
	Locational and sectoral modeling	1	[STEN93, 94a, b]	1993
	Location planning in distribution	1	[DEMM92]	1992
R & D	Learning models of consumer choice	2	[GREE87], [OLIV93]	1987 1993

Table F1.2.2. EAs in application-oriented research in management science. The third column, headed 'No', indicates the number of projects (continued).

General topic	Research application	No	References	Earliest known
Inventory	Optimizing a Wagner–Whitin model	1	[SCHO76]	1976
	Optimizing decision variables of a stochastic inventory simulation	1	[BIET94], [NISS94a, 95a, b, c]	1994
	Identifying economic order quantities	1	[STOC93]	1993
	Multicriterion inventory classification	1	[GÜVE95]	1995
	Scheduling transportation vehicles in large warehouses	1	[SCHÖ94]	1994
Production	Dynamic multiproduct, multistage lotsizing problem	1	[HELB93]	1992
	GA-based process planning model	1	[AWAD95]	1995
	Minimizing total intercell and intracell moves in cellular manufacturing	1	[GUPT92]	1992
	Line balancing	5	[ANDE90], [SCHÜ92], [BRÄH92], [WEBE93], [ANDE94a, b], [GEHR94], [LEU94], [TSUJ95a, b], [WATA95]	1990–1995
	Knowledge base refinement and rule-based simulation for an automated transportation system	1	[TERA94]	1994
	Short-term production scheduling	2	[MICH95], [KURB95a, b]	1995
	Lotsizing and scheduling	1	[GRÜN95]	1995
	Batch sequencing problem	1	[JORD94]	1994
	Parameter optimization of a simulation model for production planning	2	[AYTU94], [CLAU95, 96]	1994 1995
	Optimization tools for intelligent manufacturing systems	1	[WELL94]	1994
	Flowshop scheduling	17	[ABLA79, 87], [WERN84, 88], [BADA91], [CART91, 93b, 94], [REEV91, 92a, b, 95], [RUBI92], [STÖP92], [BIER92a, 94, 95], [BAC93], [MULK93], [CAI94], [ISHI94], [MURA94], [CHEN95a], [FICH95], [HADI95a, b], [SANG95], [STAW95]	1979–1995
	Job shop scheduling	53	[DAVI85], [HILL87, 88, 89a, b, 90], [LIEP87], [BIEG90], [HÖNE90], [KHUR90], [BAGC91], [FALK91a], [HUSB91a, b, 92, 93, 94], [KANE91], [NAKA91, 94], [NISH91], [BEAN92a], [BRUN92, 93b, 94a, b], [DORN92, 93, 95], [MORI92], [PARE92a, b, 93], [PESC92, 93], [STOR92, 93], [TAMA92], [YAMA92a], [BIER93], [CLAU93], [DAGL93, 95], [DAVI93a], [FANG93], [GEUR93], [GUPT93a, b], [JONE93], [KOPF93a], [UCKU93], [VOLT93], [APPE94], [ATLA94], [DAVE94], [GEN94a, b], [KIM94a], [LEE94], [MATT94], [PALM94b], [SHEN94], [SOAR94], [TUSO94a, b], [CHAN95], [CHEN95b], [CHOI95], [CROC95], [JÓZE95], [KIM95], [KOBA95], [LEE95b, c], [LIM95], [MCMA95], [MESM95], [NORM95], [PARK95c, d, e], [ROGN95], [RUBI95], [SZAK95], [MATT96], [OMAN96]	1985–1996

Table F1.2.2. EAs in application-oriented research in management science. The third column, headed 'No', indicates the number of projects (continued).

General topic	Research application	No	References	Earliest known
Production (continued)	Job shop group scheduling	1	[MAO95]	1995
	Discovering manufacturing control strategies	1	[BOWD92, 95]	1992
	Two-stage production scheduling problem	1	[QUIN95]	1995
	Scheduling in flexible manufacturing	3	[HOLS93], [FUJI95], [LIAN95]	1993–1995
	Scheduling in a flowline-based cell	1	[JAGA94]	1994
	Optimizing a material flow system	1	[NOCH86, 90]	1986
	Buffer optimization in assembly system	1	[BULG95]	1995
	Evolutionary parameterization of a lotsizing heuristic	1	[SCHW94]	1994
	Optimal network partition and Kanban allocation in just-in-time production	1	[ETTL95], [SCHW95, 96]	1995
	Job scheduling in electrical assembly	1	[DEO95]	1995
	Routing in manufacturing	1	[STAW95]	1995
	Scheduling and resource management in a textile factory	1	[FILI92, 93, 95]	1992
	Scheduling in steel flamecutting	1	[MURR96]	1996
	Scheduling the production of chilled ready meals	1	[SHAW96]	1996
	Sequencing jobs in car industry	2	[DEGE95], [WARW95]	1995
	Scheduling maintenance tasks	1	[GREE94]	1994
	Scheduling problem in a plastics forming plant	1	[TAMA93]	1993
	Scheduling problem in hot-rolling process	2	[TAMA94a], [MAYR95]	1994 1995
	Parallel machine tool scheduling	1	[NORM95]	1995
	Open shop scheduling	1	[FANG93]	1993
	Lotsizing and sequencing in printed circuit board manufacturing	1	[LEE93]	1993
	Decentral production scheduling	1	[WIEN94]	1994
	General production scheduling	5	[FUKU93], [WEIN93], [BLUM94], [PESC94], [STAC94]	1993–1994
	Open-pit design and scheduling	1	[DENB94a, b]	1994
	Underground mine scheduling	1	[DENB95]	1995
	Scheduling solvent production	1	[IGNI91, 93]	1991
	Maintenance scheduling	1	[KIM94b]	1994
	Machine component grouping	1	[VENU92]	1992

Table F1.2.2. EAs in application-oriented research in management science. The third column, headed 'No', indicates the number of projects (continued).

General topic	Research application	No	References	Earliest known
Production (continued)	Design of cost-minimal cable networks for production appliances	1	[SCHI81]	1981
	Grouping parts	1	[STAW95]	1995
	Ordering problem in an assembly process with constant use of parts	1	[AKAT94], [SANN94]	1994
	Information resource matrix for intelligent manufacturing	1	[KULK92]	1992
	Classification of engineering design for later reuse	1	[CAUD92]	1992
	Industrial bin packing problems	1	[FALK92, 94a, b, 95]	1992
	Two-dimensional guillotine cutting problem	1	[PARA95]	1995
	Best-cut bar problem	1	[ORDI95]	1995
Distribution	Vehicle routing	6	[THAN91a, b, 93a, b, c, 95], [LONT92], [MAZI92, 93], [BLAN93], [HIRZ92], [PANK93], [KOPF93b, 94], [UCHI94], [SCHL96]	1991–1996
	Vehicle fleet scheduling	1	[STEF95]	1995
	Transportation problems	4	[CADE94], [YANG94], [GEN94c, 95], [IDA95], [MICH96]	1989–1994
	Minimization of freight rate in commercial road transportation	1	[KOPF92]	1992
	Pallet packing/stacking in trucks	1	[JULI92, 93]	1992
	Assigning customer tours to trucks	1	[SCHR91], [BORK92, 93]	1991
	Optimizing distribution networks	1	[CAST95a, b]	1995
	Two-stage distribution problem	1	[BRAU93]	1993
Strategic management and control	Forecasting of company profit	1	[THOM86]	1986
	Project planning	2	[CHEN94b], [HART96]	1994 1996
	Calculation of budget models	1	[SPUT84]	1984
	Resource-constrained scheduling in project management	1	[LEON95]	1995
	Business system planning	1	[KNOL93]	1993
	Dynamic solutions to a strategic market game	1	[BOND94]	1994
	Portfolio optimization	1	[ABLA95b]	1995
Organization	Evolution of organizational forms under environmental selection	1	[BRUD92a, 93]	1992
	The relationship between organizational structure and ability to adapt	1	[MARE92a, b]	1992

Table F1.2.2. EAs in application-oriented research in management science. The third column, headed 'No', indicates the number of projects (continued).

General topic	Research application	No	References	Earliest known
Timetabling	Timetable problems	15	[HENS86], [ABRA91], [FANG92], [PAEC93, 94a, 95], [BURK94, 95a, b], [KINN94], [ROSS94c, 95], [FUKU95], [JACK95], [JUNG95], [MACD95], [AVIL95], [ERBE95], [FORS95], [QUEI95], [CILI96]	1986–1996
Trade	Sales forecasting for a newspaper	1	[SCHÖ93]	1993
	Selecting competitive products as part of product market analysis	1	[BALA92]	1992
	Market segmentation (deriving product market structures)	1	[HURL94, 95]	1994
	Feature selection for analyzing the characteristics of consumer goods	1	[TERA95]	1995
	Price and quantity decisions in oligopolistic markets	1	[DOSI93]	1993
	Site location of retail stores	1	[HURL94, 95]	1994
	Solving multistage location problems	1	[SCHÜ95a, b, c]	1995
	Evolution of trade strategies	1	[LENT94]	1994
	Bargaining by artificial agents	1	[DWOR96]	1996
	Analyzing efficient market hypothesis	1	[CHEN96]	1996
	Determining good pricing strategies in an oligopolistic market	1	[MARK89, 92a, b, 95]	1989
Financial services	Bankruptcy prediction	1	[SIKO92]	1992
	Loan default prediction	1	[SIKO92]	1992
	Time-series prediction	4	[GERR93], [EGLI94], [BORA95], [ROBI95]	1993–1995
	Commercial loan risk classification	1	[SIKO92]	1992
	Building classification rules for credit scoring	1	[MACK94, 95a]	1994
	Credit card attrition problem	1	[TUFT93]	1993
	Predicting horse races	1	[PERR94]	1994
	Financial analysis	1	[SING94]	1994
	Stock market forecaster	2	[MOOR94], [WARR94]	1994
	Filtering insurance applications	1	[GAMM91]	1991
	Investment portfolio selection	1	[ARNO93], [LORA95]	1993
	Stock market simulation	2	[ARTH91b], [TAYL95]	1991 1995
	Trading models evolution	1	[OUSS96]	1996

Table F1.2.2. EAs in application-oriented research in management science. The third column, headed 'No', indicates the number of projects (continued).

General topic	Research application	No	References	Earliest
Financial services (continued)	Economic modeling	1	[KOZA95]	1995
	Modeling of money markets by adaptive agents	1	[BOEH94]	1994
	Learning strategies in a multiagent stock market simulation	1	[LEBA94]	1994
	Trading automata in a computerized double auction market	5	[ANDR91, 95], [ARTH91a, 92], [MARG91, 92], [HOLL92b], [NOTT92], [ANDR94]	1990–1993
	Optimized stock investment	1	[EDDE95]	1995
	Evolutionary simulation of asset trading strategies	1	[RIEC94]	1994
	Genetic rule induction for financial decision making	2	[ALLE93], [GOON94, 95b]	1993 1994
	Neurogenetic approach to trading strategies	1	[KLIM92]	1992
	Determining parameters of business timescale (analyzing price history)	1	[DACO93]	1993
	Discovering currency investment strategies	2	[BAUE92, 94a, b, 95], [CHOP95], [PICT95]	1992 1995
	Analyzing the currency market	1	[BELT93a, b]	1993
	Negotiation support tool	1	[MATW91]	1991
Energy management	Finding multiple load flow solutions in electrical power networks	1	[YIN91, 94]	1991
	Clustering of power networks	1	[DING92]	1992
	Forecasting natural gas demand for an energy supplier	1	[SCHO93]	1993
	Unit commitment problem, generator scheduling	5	[DASG93a, b], [SHEB94],[KAZA95], [WONG95a, b, c, 96], [ORER96]	1993–1996
	Optimal arrangement of fresh and burnt nuclear fuel	1	[HEIS94a, 94b]	1994
	Fuel cycle optimization	1	[POON90, 92]	1990
Water supply systems	Designing water distribution networks	5	[CEMB79, 92], [MURP92, 93], [LOHB93], [WALT93a, b, c, 94, 95, 96], [SIMP94a, b], [DAVI95], [SAVI94a, b, 95a, c]	1992–1994
	Pressure regulation in water distribution networks to control leakage losses	1	[SAVI95e, 96]	1995
	Cost optimization of opportunity-based maintenance policies	1	[SAVI95c, d]	1995
	Pump scheduling for water supply, minimizing overall costs	1	[MACK95b]	1995
Traffic management	Optimizing train schedules to minimize passenger change times	2	[NACH93, 95a, b, c, 96], [WEZE94], [VOGE95a, b, c, d]	1993, 1994
	Scheduling underground trains	1	[HAMP81]	1981
	Scheduling urban transit systems	1	[CHAK95]	1995
	Elevator group control	1	[ALAN95]	1995

Table F1.2.2. EAs in application-oriented research in management science. The third column, headed 'No', indicates the number of projects (continued).

General topic	Research application	No	References	Earliest known
Traffic management (continued)	Controlling free flying for aircraft	1	[GERD94a, b, c, 95]	1994
	Air traffic free routing	1	[KEME95a, c], [KOK96]	1995
	Solving air traffic control conflicts	1	[ALLI93]	1993
	Partitioning air space	1	[DELA94, 95]	1994
	Airline crew scheduling	1	[CHU95]	1995
	Public transport drive scheduling	1	[WREN95]	1995
	Control of metering rates on freeway ramps	1	[MCDO95b]	1995
Personnel management	Employee staffing and scheduling	3	[EAST93], [LEVI93a, b, 95, 96], [TANO95]	1993–1995
	Talent scheduling	1	[NORD94]	1994
	Audit staff scheduling	1	[SALE94]	1994
Telecommunication	Terminal assignment in a telecommunications network	1	[ABUA94a, b]	1994
	Minimum-broadcast-time problem	1	[HOEL96a]	1996
	Finding investigator tours in telecommunication networks	1	[HOEL96b]	1996
	Analysis of call and service processing in telecommunications	1	[SINK95]	1995
	Routing in communication networks	1	[CARS95]	1995
	Design of communication networks	1	[CLIT89]	1989
Miscellaneous	General resource allocation problems	2	[SCHO76], [BEAN92a]	1976 1992
	Forecasting time series data in economic systems	1	[LEE95a]	1995
	Investigation of taxation-induced interactions in capital budgeting	1	[BERR93]	1993
	Adapting agricultural models	1	[JACU95]	1995

Table F1.2.3. EAs in other classical optimization problems.

Standard problem	References
Traveling salesman problem	[ABLA79, 87], [BRAD85], [GOLD85], [GREF85b, 87b], [HENS86], [JOG87, 91], [LIEP87, 90], [MÜHL87, 88, 91b, 92], [OLIV87], [SIRA87], [SUH87a, b], [WHIT87, 91], [FOGE88, 90, 93c, d], [HERD88, 91], [GORG89, 91a, b, c], [NAPI89], [BRAU90, 91], [JOHN90], [NYGA90, 92], [PETE90], [AMBA91, 92], [BIER91, 92b], [ESHE91], [FOX91], [GROO91], [HOFF91b], [MAND91], [RUDO91], [SENI91], [STAR91a, b, 92], [ULDE91], [BEYE92], [DAVI92], [MATH92], [MOSC92], [YAMA92b], [BAC93], [FOGA93], [HOMA93], [KIDO93], [NETT93], [PRIN93], [STAN93], [SYSW93], [TSUT93], [YANG93a], [BUI94a], [CHEN94a], [DARW94], [DZUB94], [EIBE94a], [TAMA94b], [TANG94], [TATE94], [VALE94], [YOSH94], [ABBA95], [BIAN95], [COTT95], [CRAI95], [JULS95], [ROBB95], [KURE96], [POTV96]

Table F1.2.3. EAs in other classical optimization problems (continued).

Standard problem	References
Iterated games (mostly prisoner's dilemma)	[ADAC87, 91], [AXEL87, 88], [FUJI87], [MARK89], [MILL89], [MATS90], [FOGE91, 92a, 93a, 94, 95a, b], [LIND91], [MÜHL91c], [BRUD92b], [CHAT92], [KOZA92a, b], [STAN93], [SERE94], [DAWI95], [SIEG95], [BURN95], [DARW95], [HART95], [HAO95], [YAO95b], [HO96], [JULS96], [MICH96]
Bin packing	[FOUR85], [SMIT85, 92b], [DAVI90, 92], [KRÖG91, 92], [FALK92, 94a, b, 95], [CORC93], [REEV93], [HINT94], [JAKO94a, b], [KHUR95]
Steiner problem	[HESS89], [GERR91], [HESS91], [OSTE92, 94], [JULS93], [KAPS93], [KAPS94], [ESBE95], [VASI95]
Set covering	[REPP85], [LIEP87, 90, 91], [SEN93], [SEKH93], [BEAS94], [CORN95], [BÄCK96c]
Quadratic assignment problem	[COHO86], [BROW89], [MÜHL89, 90, 91a], [LI90], [HUNT91], [MANI91, 95], [BEAN92a], [COLO92b], [LI92], [NISS92, 93a, 94a, c, d, e], [POON92], [FALC93], [TATE95], [YIP93, 94], [BUI94b], [FLEU94], [KELL94], [MARE94a, b, 95]
Assignment problems	[CART93a], [LEVI93c]
Knapsack problems	[HENS86], [GOLD87], [SMIT87, 92a], [DASG92], [GORD93], [THIE93], [KHUR94a], [MICH94], [NG95], [BÄCK96b]
Partitioning problems	[LASZ90, 91], [COHO91a, b], [COLL91], [HULI91, 92], [JONE91], [DRIE92], [MARU92, 93], [MÜHL92], [LEVI93a, 95, 96], [INAY94], [KHUR94d], [HÖHN95], [KAHN95], [MENO95], [BÄCK96b]
Scheduling (general)	[SANN88], [HOU90, 92], [LAWT92], [SMIT92c], [KIDW93], [ADIT94a, b], [ALI94], [ANDE94a], [CHAN94b], [CORC94], [GONZ94], [HOU94], [KHUR94d], [PICO94], [SCHW94], [SEIB94], [WAH95]
Graph coloring	[DAVI90, 91], [EIBE94a], [COST95], [FLEU95, 96]
Minimum vertex cover	[KHUR94b, c]
Miscellaneous graph problems	[BÄCK94, 96b], [PALM94a], [ABUA95a, b, 96], [PIGG95]
Mapping problems	[MANS91], [NEUH91], [ANSA92], [SOUL96]
Maximum clique problem	[BAZG95], [BUI95], [FLEU95b], [PARK95a, b]
Maximum-flow problem	[MUNA93]
General integer programming	[ABLA79], [BEAN92b], [HADJ92], [RUDO94]
Satisfiability problem	[JONG89], [FLEU95b], [HAO95], [PARK95a, b]
Routing problems	[LIEN94a, b], [MARI94]
Subset sum problem	[KHUR94d]
Query optimization	[YANG93b], [STIL96]
Task allocation	[FALC95]
Load balancing in a database	[ALBA95]

Appendix B. Extensive bibliography

[ABBA95] Abbatista F 1995 Travelling salesman problem solved with GA and ant system *Genetic Algorithms Digest (E-mail list)* 9 (41) 7.8.1995

[ABLA79] Ablay P 1979 Optimieren mit Evolutionsstrategien. Reihenfolgeprobleme, nichtlineare und ganzzahlige Optimierung *Doctoral Dissertation* University of Heidelberg

[ABLA87] Ablay P 1987 Optimieren mit Evolutionsstrategien *Spektrum der Wissenschaft* **7** 104–15

[ABLA90] Ablay P 1990 Konstruktion kontrollierter Evolutionsstrategien zur Lösung schwieriger Optimierungsprobleme der Wirtschaft *Evolution und Evolutionsstrategien in Biologie, Technik und Gesellschaft* ed J Albertz (Wiesbaden: Freie Akademie) 2nd edn, pp 73–106

[ABLA91] Ablay P 1991 Ten theses regarding the design of controlled evolutionary strategies. In: [BECK91] pp 457–481

[ABLA92] Ablay P 1992 1. Optimal placement of railtrack reconstruction sites; 2. Scheduling of patients in a hospital; 3. Scheduling cleaning personnel for trains; personal communication with P Ablay

[ABLA95a] Ablay P 1995 Evolutionäre Strategien im marktwirtschaftlichen Einsatz *Business Paper* (Gräfelfing/Munich: Ablay Optimierung)

[ABLA95b] Ablay P 1995 Portfolio-optimization in the context of deciding between projects of different requirements and expected cash-flows, personal communication

[ABRA91] Abramson D and Abela J 1991 A parallel genetic algorithm for solving the school timetabling problem *Technical Report* TR-DB-91-02 (RMIT TR 118 105 R), Division of Information Technology, Macquarie Centre, North Ryde, NSW, Australia

[ABRA93a] Abramson D 1993 Scheduling aircraft landing times, personal communication

[ABRA93b] Abramson D, Mills G and Perkins S 1993 Parallelisation of a genetic algorithm for the computation of efficient train schedules *Proc. 1993 Parallel Computing and Transputers Conf.* (Amsterdam: IOS) pp 1–11

[ABRA94] Abramson D, Mills G and Perkins S 1994 Parallelism of a genetic algorithm for the computation of efficient train schedules *Parallel Computing and Transputers* ed D Arnold, R Christie, J Day and P Roe (Amsterdam: IOS) pp 139–49

[ABUA94a] Abuali F N, Schoenefeld D A and Wainwright R L 1994 Terminal assingment in a communications network using genetic algorithms *Proc. ACM Computer Science Conf. (CSC'94) (Phoenix, AZ, 1994)* (New York: ACM) pp 74–81

[ABUA94b] Abuali F N, Schoenefeld D A and Wainwright R L 1994 Designing telecommunications networks using genetic algorithms and probabilistic minimum spanning trees *Proc. 1994 ACM/SIGAPP Symp. on Applied Computing* (New York: ACM) pp 242–46

[ABUA95a] Abuali F N, Wainwright R L and Schoenefeld D A 1995 Determinant factorization and cycle basis: encoding scheme for the representation of spanning trees on incomplete graphs *Proc. 1995 ACM/SIGAPP Symp. on Applied Computing* (New York: ACM) pp 305–12

[ABUA95b] Abuali F N, Wainwright R L and Schoenefeld D A 1995 Determinant factorization: a new encoding scheme for spanning trees applied to the probabilistic minimum spanning tree problem. In: [ESHE95] pp 470–77

[ABUA96] Abuali F N, Wainwright R L and Schoenefeld D A 1996 Solving the subset interconnection design problem using genetic algorithms *Proc. 1996 ACM/SIGAPP Symp. on Applied Computing* (New York: ACM) pp 299–304

[ADAC87] Adachi N 1987 Framework of mutation model for evolution in the ecological model game world *IIAS-SIS Research Report* 74, Numazu, Japan

[ADAC91] Adachi N and Matsuo K 1991 Ecological dynamics under different selection rules in distributed and iterated prisoner's dilemma game. In: [SCHW91] pp 388–94

[ADER85] Adermann H-J 1985 Treatment of integral contingent conditions in optimum power plant commitment *Applied Optimization Techniques in Energy Problems* ed H J Wacker (Stuttgart: Teubner) pp 1–18

[ADIT94a] Aditya S K, Bayoumi M and Lursinap C 1994 Genetic algorithm for near optimal scheduling and allocation in high level synthesis. In: [KUNZ94] pp 85–6

[ADIT94b] Aditya S K, Bayoumi M and Lursinap C 1994 Genetic algorithm for near optimal scheduling and allocation in high level synthesis. In: [HOPF94] pp 91–7

[AKAT94] Akatsuta N, Sannomiya N and Iima H 1994 Genetic algorithm approach to a production ordering problem in an assembly process with constant use of parts *Int. J. Systems Sci.* **25** 1461

[ALAN95] Alander J, Ylinen J and Tyni T 1995 Elevator group control using distributed genetic algorithms. In: [PEAR95] pp 400–3

[ALBA95] Alba E, Aldana J F and Troya J M 1995 A genetic algorithm for load balancing in parallel query evaluation for deductive relational databases. In: [PEAR95] pp 479–82

[ALBR93] Albrecht R F, Reeves C R and Steele N C (ed) 1993 *Artificial Neural Nets and Genetic Algorithms (Innsbruck, April 13–16 1993)* (Berlin: Springer)

[ALI94] Ali S, Sait S M and Benten M S T 1994 GSA: scheduling and allocation using genetic algorithm *Proc. Eur. Design Automation Conf.* (Los Alamitos, CA: IEEE Computer Society) 84–9

[ALLE93] Allen F and Karjalainen R 1993 Using genetic algorithms to find technical trading rules *Technical Report* 20-93, Rodney L White Center for Financial Research, The Wharton School, University of Pensylvania

[ALLI93] Alliot J M, Gruber H, Joly G and Schoenauer M 1993 Genetic algorithms for solving air traffic control conflicts *Proc. 9th Int. Conf. on Artificial Intelligence for Applications (Orlando, FL, March 1–5 1993)* (Los Alamitos, CA: IEEE Computer Society) pp 338–44

[AMBA91] Ambati B K, Ambati J and Mokhtar M M 1991 Heuristic combinatorial optimization by simulated Darwinian evolution: a polynomial time algorithm for the traveling salesman problem *Biol. Cybern.* **65** 31–5

[AMBA92] Ambati B K, Ambati J and Mokhtar M M 1992 An $O(n \log n)$-time genetic algorithm for the traveling salesman problem. In: [FOGE92b] pp 134–40

[ANDE90] Anderson E J and Ferris M C 1990 A genetic algorithm for the assembly line balancing problem *Proc. Integer Programming/Combinatorial Optimization Conf.* (Ontario: University of Waterloo Press) pp 7–18

[ANDE94a] Andersen B 1994 Tuning computer CPU scheduling algorithms. In: [SEBA94] pp 316–23

[ANDE94b] Anderson E J and Ferris M C 1994 Genetic algorithms for combinatorial optimization: the assembly line balancing problem *ORSA J. Comput.* **6** 161–73

[ANDE94c] Anderson E J and Ferris M C 1994 Genetic algorithms for combitorial optimization: the assembly line balancing problem *Operations Res./Management Sci.* **34** 523–5

[ANDR91] Andreoni J and Miller J H 1991 Auctions with adaptive artificially intelligent agents *Working Paper* 91-01-004, Santa Fe Institute

[ANDR94] Andrews M and Prager R 1994 Genetic programming for the acquisition of double auction market strategies *Advances in Genetic Programming* ed K E Kinnear Jr (Cambridge, MA: MIT Press) pp 355–368

[ANDR95] Andreoni J and Miller J H 1995 Auctions with artificial adaptive agents *Games Econ. Behavior* **10** 39–64

[ANSA92] Ansari N, Chen M H and Hou E S H 1992 A genetic algorithm for point pattern matching. In: [SOUC92] pp 353–71

[APPE94] Appelrath H J and Bruns R 1994 Genetische Algorithmen zur Lösung von Ablaufplanungsproblemen *Fuzzy Logik. Theorie und Praxis* ed B Reusch (Berlin: Springer) pp 25–32

[ARNO93] Arnone S, Loraschi A and Tettamanzi A 1993 A genetic approach to portfolio selection *Neural Network World* **3** 597

[ARTH91a] Arthur B W 1991 Designing economic agents that act like human agents: a behavioral approach to bounded rationality *Am. Econ. Rev.* **81** 353–9

[ARTH91b] Arthur B, Holland J, Palmer R and Tayler P 1991 Using genetic algorithms to model the stock market *Proc. Forecasting and Optimization in Financial Services Conf.* (London: IBC Technical Services)

[ARTH92] Arthur B W 1992 On learning and adaption in the economy *Working Paper* 92-07-038, Santa Fe Institute

[ATLA94] Atlan L, Bonnet J and Naillon M 1993 Learning distributed reactive strategies by genetic programming for the general job shop problem *Proc. 7th Ann. Florida Artificial Intelligence Research Symp.* (New York: IEEE)

[ATMA92] Atmar W 1992 On the rules and nature of simulated evolutionary programming. In: [FOGE92b] pp 17–26

[AVIL95] Avila I 1995 GA to construct school timetables *Timetabling Problems List (E-Mail List)* 15.6.1995

[AWAD95] Awadh B, Sepehri N and Hawaleshka O 1995 A computer-aided process planning model based on genetic algorithms *Comput. Operations Res.* **22** 841–56

[AXEL87] Axelrod R 1987 The evolution of strategies in the iterated prisoner's dilemma *Genetic Algorithms and Simulated Annealing* ed L Davis (San Mateo, CA: Morgan Kaufmann) pp 32–41

[AXEL88] Axelrod R and Dion D 1988 The further evolution of cooperation *Science* **242** 1385–90

[AXMA94a] Axmann J K and Turan R 1994 Parallele adaptive evolutionäre Algorithmen zur Brennelement-Einsatzplanung *Research Paper* Department of Aeronautics and Reactor Technology, University of Braunschweig

[AXMA94b] Axmann J K 1994 Optimierung von DWR-Nachladungen mit adaptiven Evolutionsalgorithmen auf Parallelrechnern *Proc. Jahrestagung Kerntechnik (Stuttgart, 1994)* (Bonn: INFORUM)

[AYTU94] Aytug H, Koehler G J and Snowdon J L 1994 Genetic learning of dynamic scheduling within a simulation environment *Comput. Operations Res.* **21** 909–25

[BAC93] Bac F Q and Perov V L 1993 New evolutionary genetic algorithms for NP-complete combinatorial optimization problems *Biol. Cybern.* **69** 229–34

[BÄCK92] Bäck T, Hoffmeister F and Schwefel H-P (ed) 1992 Applications of evolutionary algorithms *Technical Report* SYS-2/92, Computer Science Department, University of Dortmund, Germany

[BÄCK94] Bäck T and Khuri S 1994 An evolutionary heuristic for the maximum independent set problem *Proc. 1st IEEE Conf. on Evolutionary Computation* (Piscataway, NJ: IEEE) pp 531–5

[BÄCK95] Bäck T, Beielstein T, Naujoks B and Heistermann J 1995 Evolutionary algorithms for the optimization of simulation models using PVM *EuroPVM '95: Second Eur. PVM Users' Group Meeting (Hermes, Paris)* ed J Dongarra, M Gengler, B Tourancheau and X Vigouroux pp 277–82

[BÄCK96a] Bäck T, Heistermann J, Kappler C and Zamparelli M 1996 Evolutionary algorithms support refueling of pressurized water reactors *Proc. 3rd IEEE Conf. on Evolutionary Computation* (Piscataway, NJ: IEEE)

[BÄCK96b] Bäck T and Schütz M 1996 Intelligent mutation rate control in canonical genetic algorithms, foundations of intelligent systems *Proc. 9th Int. Symp. on Methodologies for Intelligent Systems (Lecture Notes in Artificial Intelligence 1079)* ed Z Ras (Berlin: Springer) pp 158–67

[BÄCK96c] Bäck T, Schütz M and Khuri S 1996 A comparative study of a penalty function, a repair heuristic, and stochastic operators with the set-covering problem *Evolution Artificielle (Lecture Notes in Computer Science 1063)* ed M Schoenhauer M and E Ronald (Berlin: Springer) pp 320–32

[BADA91] Badami V S and Parks C M 1991 A classifier based approach to flow shop scheduling *Comput. Ind. Eng.* **21** 329–33

[BAGC91] Bagchi S, Uckun S, Miyabe Y and Kawamura K 1991 Exploring problem-specific recombination operators for job shop scheduling. In: [BELE91] pp 10–17

[BAIT95] Baita F, Mason F, Poloni C and Ukovich W 1995 Genetic algorithm with redundancies for the vehicle scheduling problem. In: [BIET95a] pp 341–54

[BAKE91] Baker E K and Schaffer J R 1991 Solution improvement heuristics for the vehicle routing problems with time window constraints *Am. J. Math. Management Sci.* **6** 261–300

[BALA92] Balakrishman P V S and Jacob V S 1992 A genetic algorithm based decision support system for optional product design, paper presented at: TIMS Ann. Conf., London Business School, London, 15–17 July 1992

[BALU95] Baluja S 1995 An empirical comparison of seven iterative and evolutionary function optimization heuristics *Technical Report* CMU-CS-95-193, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA

[BAUE92] Bauer R J and Liepins G L 1992 Genetic algorithms and computerized trading strategies *Expert Systems in Finance* ed D E O'Leary and P R Watkins (Amsterdam: Elsevier) pp 89–100

[BAUE94a] Bauer R J 1994 *Genetic Algorithms and Investment Strategies* (New York: Wiley)

[BAUE94b] Bauer R J 1994 An introduction to genetic algorithms: a mutual fund screening example *Neurovest J.* **4** 16–19

[BAUE95] Bauer R J 1995 Genetic algorithms and the management of exchange rate risk. In: [BIET95a] pp 253–63

[BAZG95] Bazgan C and Lucian H 1995 A genetic algorithm for the maximal clique problem. In: [PEAR95] pp 499–502

[BEAN92a] Bean J C 1992 Genetics and random keys for sequencing and optimization *Technical Report* 92-43, Department of Industrial and Operations Engineering, The University of Michigan, Ann Arbor, MI

[BEAN92b] Bean J C and Hadj-Alouane A B 1992 A dual genetic algorithm for bounded integer programs *Technical Report* 92-53, Department of Industrial and Operations Research, University of Michigan, Ann Arbor, MI

[BEAS90] Beasley J E 1990 OR-library: distributing test problems by electronic mail *J. Operations Res. Soc.* **41** 1069–72

[BEAS94] Beasley J E and Chu P C 1994 A genetic algorithm for the set covering problem *Genetic Algorithms Digest (E-mail List)* 8 (38) 5.10.1994

[BECK91] Becker J D, Eisele I and Mündemann F W (ed) 1991 *Parallelism, Learning, Evolution, Proc. Workshop on Evolutionary Models and Strategies (Neubiberg, 10–11 March 1989), and Workshop on Parallel Processing: Logic, Organization, and Technology—WOPPLOT 89 (Wildbad Kreuth, 24–28 July 1989)* (Berlin: Springer)

[BELE91] Belew R K and Becker L B (ed) 1991 *Proc. 4th Int. Conf. on Genetic Algorithms (UCSD, San Diego, July 13–16 1991)* (San Mateo, CA: Morgan Kaufmann)

[BELT93a] Beltrametti L, Marengo L and Tamborini R 1993 A learning experiment with classifier system: the determinants of the dollar–mark exchange rate *Discussion Paper* 5/93, Department of Economics, University of Trento, Italy

[BELT93b] Beltratti A and Margarita S 1993 Evolution of trading strategies among heterogeneous artificial economic agents *From Animals to Animats, Proc. 2nd Int. Conf. on Simulation of Adaptive Behavior (SAB92) (Honolulu, December 7–11 1992)* ed H Roitblat H, J A Meyer and S W Wilson (Cambridge, MA: MIT Press) pp 494–501

[BERE91] Berens W 1991 *Beurteilung von Heuristiken* (Wiesbaden: Gabler)

[BERR93] Berry R H and Smith G D 1993 Using a genetic algorithm to investigate taxation induced interactions in capital budgeting. In: [ALBR93] pp 567–74

[BEYE92] Beyer H-G 1992 Some aspects of the 'evolution strategy' for solving TSP-like optimization problems appearing at the design studies of a 0.5 TeV e^+e^- linear collider. In: [MÄNN92] pp 361–70

[BIAN95] Bianchini R, Brown C M, Cierniak M and Meira W 1995 Combining distributed populations and periodic centralized selections in coarse-grain parallel genetic algorithms. In: [PEAR95] pp 483–6

[BIEG90] Biegel J E and Davern J J 1990 Genetic algorithms and job shop scheduling *Comput. Ind. Eng.* **19** 81–91

[BIER91] Bierwirth C, Mattfeld D C and Stöppler S 1991 Pseudo-parallelity and distributed programming under UNIX system V *Proc. Conf. on Parallel Algorithms and Transputers for Optimization* ed M Grauer and D B Pressmar (Berlin: Springer) pp 35–44

[BIER92a] Bierwirth C 1992 Optimierung der Ablaufplanung einer Fliessfertigung mit Parallelen Genetischen Algorithmen *Doctoral Dissertation* Department of Economics, University of Bremen

[BIER92b] Bierwirth C and Mattfeld D C 1992 Implementierung eines Parallelen Genetischen Algorithmus in einem verteilten System, paper presented at: PECOR Workshop of the DGOR, Düsseldorf, 4 December

[BIER93] Bierwirth C, Kopfer H, Mattfeld D and Utecht T 1993 Genetische Algorithmen und das Problem der Maschinenbelegung. Eine Übersicht und ein neuer Ansatz *Technical Report* 3, Department of Economics, University of Bremen

[BIER94] Bierwirth C 1994 *Flowshop Scheduling mit Parallelen Genetischen Algorithmen—Eine Problemorientierte Analyse genetischer Suchstrategien* (Wiesbaden: Vieweg/DUV)

[BIER95] Bierwirth C 1995 A generalized permutation approach to job shop scheduling with genetic algorithms *OR Spektrum* **17** 87–92

[BIET94] Biethahn J and Nissen V 1994 Combinations of simulation and evolutionary algorithms in management science and economics *Ann. OR* **52** 183–208

[BIET95a] Biethahn J and Nissen V (ed) 1995 *Evolutionary Algorithms in Management Applications* (Berlin: Springer)

[BIET95b] Biethahn J (ed) 1995 Simulation als betriebliche Entscheidungshilfe: Neuere Werkzeuge und Anwendungen aus der Praxis *Proc. 5th Simulation Symp. (Braunlage, 13–15 March 1995)*

[BLAN93] Blanton J L and Wainwright R L 1993 Multiple vehicle routing with time and capacity constraints using genetic algorithms. In: [FORR93] pp 452–9

[BLUM94] Blume C 1994 Planning and optimization of scheduling in industrial production by genetic algorithms and evolutionary strategy *Methodologies, Techniques and Tools for Design Development, Am. Soc. Mech. E., Petroleum Div.* **64** 427–33

[BOEH94] Boehme F 1994 Adaptive coordination mechanism in the economic modelling of money markets. In: [HILL94] pp 179–99

[BOND94] Bond G, Liu J and Shubik M 1994 Dynamic solutions to a strategic market game: analysis, programming and a genetic algorithm approach. In: [HILL94] pp 84–111

[BORA95] Borasky M E 1995 Financial forecasting, personal communication

[BORK92] Borkowski V 1992 Entwicklung eines Genetischen Algorithmus zur Fahrzeugeinsatzplanung im Werksverkehr *Diploma Thesis* Department of Economics, University of Hagen, Germany

[BORK93] Borkowski V 1993 Vergleich zwischen einem Expertensystem und alternativen Entscheidungsunterstützungs-Methoden in der Vertriebslogistik *Doctoral Dissertation* University of Erlangen-Nürnberg *Technical Reports of the Computer Science Department* vol 26, No 6, Erlangen

[BOWD92] Bowden R O 1992 Genetic algorithm based machine learning applied to the dynamic routing of discrete parts *Doctoral Dissertation* Department of Industrial Engineering, Mississippi State University, MS

[BOWD95] Bowden R and Bullington S F 1995 An evolutionary algorithm for discovering manufacturing control strategies. In: [BIET95a] pp 124–38

[BRAD85] Brady R M 1985 Optimization strategies gleaned from biological evolution *Nature* **317** 804–6

[BRÄH92] Brähler U 1992 Abstimmung von Montagelinien mit einem Genetischen Algorithmus *Diploma Thesis* Department of Economics, University of Hagen, Germany

[BRAN89] Brandeau M L and Chiu S S 1989 An overview of representative problems in location research *Management Sci.* **35** 645–74

[BRAN95] Branke J, Kohlmorgen U, Schmeck H and Veith H 1995 Steuerung einer Heuristik zur Losgrössenplanung unter Kapazitätsrestriktionen mit Hilfe eines parallelen genetischen Algorithmus. In: [KUHL95] pp 21–31

[BRAU90] Braun H 1990 Massiv parallele Algorithmen für kombinatorische Optimierungsprobleme und ihre Implementierung auf einem Parallelrechner *Doctoral Dissertation* University of Karlsruhe (TH)

[BRAU91] Braun H 1991 On solving travelling salesman problems by genetic algorithms. In: [SCHW91] pp 129–33

[BRAU93] Braun J 1993 Entwicklung eines Genetischen Algorithmus zur Lösung eines zweistufigen Distributionsproblems *Diploma Thesis* Department of Economics, University of Hagen, Germany

[BROE95a] Broekmeulen R A C M 1995 Facility management of distribution centres for vegetables and fruits. In: [KUHL95] pp 32–9

[BROE95b] Broekmeulen R A C M 1995 Facility management of distribution centres for vegetables and fruits. In: [BIET95a] pp 199–210

[BROW89] Brown D E, Huntley C L and Spillane A R 1989 A parallel genetic heuristic for the quadratic assignment problem. In: [SCHA89] pp 406–15

[BRUD92a] Bruderer E 1992 How organizational learning guides environmental selection *Working Paper* School of Business Administration, University of Michigan, Ann Arbor, MI

[BRUD92b] Bruderer E 1992 Strategic learning *Working Paper* University of Michigan, Ann Arbor, MI

[BRUD93] Bruderer E 1993 How strategies are learned *Doctoral Dissertation* University of Michigan, Ann Arbor, MI

[BRUN92] Bruns R 1992 Incorporation of a knowledge-based scheduling system into a genetic algorithm *Proc. 'GI-Jahrestagung' (Karlsruhe, 1992)* (Berlin: Springer)

[BRUN93a] Bruns R 1993 Direct chromosome representation and advanced genetic operators for production scheduling. In: [FORR93] pp 352–9

[BRUN93b] Bruns R 1993 Knowledge-augmented genetic algorithm for production scheduling *IJCA-93 Workshop on Knowledge-Based Production Planning, Scheduling, and Control (Chambery, 29 August 1993)* (Menlo Park, CA: AAAI) pp 49–58

[BRUN94a] Bruns R 1994 GAP—a knowledge-augmented genetic algorithm for industrial production scheduling problems. In: [KUNZ94] pp 87–8

[BRUN94b] Bruns R 1994 GAP—a knowledge-augmented genetic algorithm for industrial production scheduling problems. In: [HOPF94] pp 98–9

[BUI94a] Bui T N and Moon B R 1994 A new genetic approach for the travelling salesman problem *Proc. 1st IEEE Conf. on Evolutionary Computation (Orlando, FL, June 27–29 1994)* vol 1 (Piscataway, NJ: IEEE) pp 7–12

[BUI94b] Bui T N and Moon B R 1994 A genetic algorithm for a special class of the quadratic assignment problem *Quadratic Assignment and Related Problems (DIMACS Series in Discrete Mathematics and Theoretical Computer Science 16)* ed P M Pardalos and H Wolkowicz (Providence, RI: American Mathematical Society) pp 99–116

[BUI95] Bui T N and Eppley P H 1995 A hybrid genetic algorithm for the maximum clique problem. In: [ESHE95] pp 478–83

[BULG95] Bulgak A A, Diwan P D and Inozu B 1995 Buffer size optimization in asynchronous assembly systems using genetic algorithms *Comput. Ind. Eng.* **28** 309–22

[BURK94] Burke E, Elliman D and Weare R 1994 A university timetabling system based on graph colouring and constraint manipulation *J. Res. Comput. Education* **27** 1–18

[BURK95a] Burke E K, Elliman D G and Weare R F 1995 A hybrid genetic algorithm for highly constrained timetabling problems. In: [ESHE95] pp 605–10

[BURK95b] Burke E, Elliman D and Weare R 1995 Specialised recombinative operators for timetabling problems. In: [FOGA95c] pp 75–85

[BURN95] Burns T D 1995 GA to solve simple game theory problems *Genetic Algorithms Digest (E-mail List)* 9 (36) 22.6.1995

[BUSC91] Busch M 1991 Paretooptimale Strategien für ein PPS-System mittels Simulation *Diploma Thesis* Department of Computer Science, University of Dortmund, Germany

[CADE94] Cadenas J M and Jiménez F 1994 A genetic algorithm for the multiobjective solid transportation problem: a fuzzy approach *Proc. Int. Symp. on Automotive Technology and Automation 1994* (Croydon, UK: Automotive Automation) pp 327–34

[CAI94] Cai X and Ka-Wai L 1994 A genetic algorithm for flow shop scheduling with travel times between machines *Proc. 2nd Eur. Congress on Intelligent Techniques and Soft Computing (EUFIT'94) (Aachen, 20–23 September 1994)* vol 1 (Aachen: ELITE Foundation) pp 186–8

[CARS95a] Carse B and Fogarty T C and Munro A 1995 Adaptive distributed routing using evolutionary fuzzy control. In: [ESHE95] pp 389–96

[CARS95b] Carse B and Fogarty T C and Munro A 1995 Evolutionary learning in computational ecologies: an application to adaptive, distributed routing in communication networks. In: [FOGA95c] pp 103–16

[CART91] Cartwright H M and Mott G F 1991 Looking around: using clues from data space to guide genetic algorithm searches. In: [BELE91] pp 108–14

[CART93a] Cartwright H M and Harris S P 1993 The application of the genetic algorithm to two-dimensional strings: the source apportionment problem. In: [FORR93] p 631

[CART93b] Cartwright H M and Long R A 1993 Simultaneous optimisation of chemical flowshop sequencing and topology using genetic algorithms *Ind. Eng. Chem. Res.* **32** 2706–13

[CART94] Cartwright H M and Tuson A L 1994 Genetic algorithms and flowshop scheduling: towards the development of a real-time process control system. In: [FOGA94a] pp 277–90

[CAST95a] Castillo L and González A 1995 Fuzzy optimization of distribution networks by using genetic algorithms *Technical Report* DECSAI-95131, Department of Computer Science and Artificial Intelligence, University of Granada, Spain

[CAST95b] Castillo L and González A 1995 Optimizing the final cost in distribution networks under fuzzy restrictions *Proc. 6th Int. Fuzzy Systems Association World Congress (IFSA'95)* vol 2, pp 81–4

[CAUD92] Caudell T P 1992 Genetic algorithms as a tool for the analysis of adaptive resonance theory network training sets *COGANN-92, Int. Workshop on Combinations of Genetic Algorithms and Neural Networks* ed L Whitley and J D Schaffer (Los Alamitos, CA: IEEE Computer Society) pp 184–200

[CEMB79] Cembrowicz R G and Krauter G E 1979 Optimization of urban and regional water supply systems *Proc. Conf. on Systems Approach for Development (Cairo)* ed M A R Ghonaimy (Oxford: International Federation of Automatic Control) pp 449–54

[CEMB92] Cembrowicz R G 1992 Water supply systems optimisation for developing countries *Pipeline Systems* ed B Coulbeck and E Evans (Dordrecht: Kluwer) pp 59–76

[CHAK95] Chakroborty P, Deb K and Subrahmanyam P S 1995 Optimal scheduling of urban transit systems using genetic algorithms. *J. Transportation Eng.* **121** 544–53

[CHAN94a] Chan K C and Tansri H 1994 Study of genetic crossover operations on the facilities layout problem *Comput. Ind. Eng.* **26** 537–50

[CHAN94b] Chandrasekharam R, Vinod V V and Subramanian S 1994 Genetic algorithm for test scheduling with different objectives *Integration, VLSI J.* **17** 153–61

[CHAN95] Changshui Z and Pingfan Y 1995 A genetic algorithm of solving job-shop scheduling problem *Chinese J. Electron.* **4** 48–52

[CHAT92] Chattoe E 1992 Evolutionary models of economic agency, personal communication

[CHEN94a] Chen H and Flann N S 1994 Parallel simulated annealing and genetic algorithms: a space of hybrid methods. In: [DAVI94a] pp 428–38

[CHEN94b] Cheng R and Gen M 1994 Evolution program for resource constrained project scheduling problem *Proc. 1st IEEE Conf. on Evolutionary Computing (Orlando, FL, June 27–29 1994)* vol 2 (Piscataway, NJ: IEEE) pp 736–41

[CHEN95a] Chen C L, Vempati V S and Aljaber N 1995 An application of genetic algorithms for flow shop problem *Eur. J. Operations Res.* **80** 389–96

[CHEN95b] Cheng R W and Gen M S and Tozawa T 1995 Minmax earliness tardiness scheduling in identical parallel machine system using genetic algorithms *Comput. Ind. Eng.* **29** 513–17

[CHEN96] Chen S H and Yeh C-H 1996 Genetic programming and the efficient market hypothesis. In: [KOZA96a]

[CHOI95] Choi J Y and Lee C Y 1995 A genetic algorithm for job sequencing problems with distinct due dates and general early–tardy penalty weights *Comput. Operations Res.* **22** 857

[CHOP95] Chopard B, Oussaidène M, Pictet O V, Schirru R and Tomassini M 1995 Evolutionary algorithms for multimodal optimization in financial applications *Proc. SPP-IF Seminar, Zürich (1995)* pp 139–42

[CHU95] Chu P C and Beasley J E 1995 A genetic algorithm for the set partitioning problem *Research Paper* Management School, Imperial College, London

[CILI96] Cilingir E 1996 Timetabling. *Timetabling Problems List (E-Mail List)* 5.3.1996

[CLAU93] Claus T 1993 Genetische Simulation, paper presented at: 4th Symp. 'Simulation als betriebliche Entscheidungshilfe', Braunlage, 15–17 March 1993

[CLAU95] Claus T 1995 Objektorientierte Simulation und evolutionäre Parameteroptimierung. In: [BIET95b] pp 87–96

[CLAU96] Claus T 1996 *Objektorientierte Simulation und Genetische Algorithmen zur Produktionsplanung und -steuerung* (Frankfurt am Main: Lang)

[CLEV89] Cleveland G A and Smith S F 1989 Using genetic algorithms to schedule flow shop releases. In: [SCHA89] pp 160–9

[CLIT89] Clitherow P and Fisher G 1989 Knowledge based assistance of genetic search in large design spaces *Proc. 2nd Int. Conf. on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA/AIE 89)* (New York: ACM) pp 729–34

[COHO86] Cohoon J P and Paris W D 1986 Genetic placement *Proc. IEEE Int. Conf. on Computer Aided Design* (Digest of Technical Papers) (New York: IEEE Computer Society Press) pp 422–5

[COHO91a] Cohoon J P, Martin W N and Richards D S 1991 Genetic algorithms and punctuated equilibria. In: [SCHW91] pp 134–44

[COHO91b] Cohoon J P, Martin W N and Richards D S 1991 A multi-population genetic algorithm for solving the K-partition problem on hyper-cubes. In: [BELE91] pp 244–8

[COLL91] Collins R J and Jefferson D R 1991 Selection in massively parallel genetic algorithms. In: [BELE91] pp 249–56

[COLO91a] Colorni A, Dorigo M and Maniezzo V 1991 Genetic algorithms and highly constrained problems: the time-table case. In: [SCHW91] pp 55–9

[COLO91b] Colorni A, Dorigo M and Maniezzo V 1991 Gli algoritmi genetici e il problema dell'orario (Genetic algorithms and the problem of timetables) *Ricerca Operativa* **60** 5–31 (in Italian)

[COLO92a] Colorni A, Dorigo M and Maniezzo V 1992 Genetic algorithms: a new approach to the timetable problem *Combinatorial Optimization. New Frontiers in Theory and Practice* ed M Akgül, H W Hamacher H W and S Tüfekci (Berlin: Springer) pp 235–9

[COLO92b] Colorni A, Dorigo M and Maniezzo V 1992 ALGODESK: an experimental comparison of eight evolutionary heuristics applied to the QAP problem *Report* 92-052, Department of Electronics, Milan Polytechnic, Italy

[CONW94] Conway D G and Venkataramanan M A 1994 Genetic search and the dynamic facility layout problem *Comput. Operations Res.* **21** 955–60

[COOM87] Coombs S and Davis L 1987 Genetic algorithms and communication link speed design: constraints and operators. In: [GREF87a] pp 257–60

[CORC93] Corcoran A L and Wainwright R L 1993 A heuristic for improved genetic bin packing *Technical Report* UTULSA-MCS-93-8, University of Tulsa, OK

[CORC94] Corcoran A L and Wainwright R L 1994 A parallel island model genetic algorithm for the multiprocessor scheduling problem *Proc. 1994 ACM/SIGAPP Symp. on Applied Computing* (New York: ACM) pp 483–7

[CORN93] Corne D, Fang H L and Mellish C 1993 Solving the module exam scheduling problem with genetic algorithms *Proc. 6th Int. Conf. on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems* ed P W H Chung, G Lovegrove and M Ali (London: Gordon and Breach) pp 370–3

[CORN94] Corne D, Ross P and Fang H-L 1994 Fast practical evolutionary timetabling. In: [FOGA94a] pp 250–63

[CORN95] Corne D and Ross P 1995 Some combinatorial landscapes on which a genetic algorithm outperforms other stochastic iterative methods. In: [FOGA95c] 1–13

[COST95] Costa D, Hertz A and Dubuis O 1995 Embedding of a sequential procedure within an evolutionary algorithm for coloring problems in graphs. *J. Heuristics* **1**

[COTT95] Cotta C, Aldana J F, Nebro A J and Troya J M 1995 Hybridizing genetic algorithms with branch and bound techniques for the resolution of the TSP. In: [PEAR95] pp 277–80

[COX91] Cox L A Jr, Davis L and Qiu Y 1991 Dynamic anticipatory routing in circuit-switched telecommunications networks. In: [DAVI91] pp 124–43

[CRAI95] Craighurst R and Martin W 1995 Enhancing GA performance through crossover prohibitions based on ancestry. In: [ESHE95] pp 130–5

[CROC95] Della Croce F, Tadei R and Volta G 1995 A genetic algorithm for the job shop problem *Comput. Operations Res.* **22** 15–24

[DACO93] Dacorogna M M, Mueller U A, Nagler R J, Olsen R B and Pictet O V 1993 A geographical model for the daily and weekly seasonal volatility in the fx market. *J. Int. Money Finance* **12** 413–38

[DAGL93] Dagli C and Sittisathanchai S 1993 Genetic neuro-scheduler for job shop scheduling *Comput. Ind. Eng.* **25** 267–70

[DAGL95] Dagli C H and Sittisathanchai S 1995 Genetic neuro-scheduler—a new approach for job-shop scheduling *Int. J. Prod. Econ.* **41** 135–45

[DARW94] Darwen P J and Yao X 1994 On evolving robust strategies for iterated prisoner's dilema *Technical Report* Department of Computer Science, Australian Defence Force Academy, University College, University of New South Wales, Canberra

[DARW95] Darwen P J and Yao X 1995 On evolving robust strategies for iterated prisoner's dilema. In: [YAO95a] pp 276–92

[DASG92] Dasgupta D and McGregor D R 1992 Nonstationary function optimization using the structured genetic algorithm. In: [MÄNN92] pp 145–54

[DASG93a] Dasgupta D 1993 Unit commitment in thermal power generation using genetic algorithms *Proc. 6th Int. Conf. on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems* ed P W H Chung, G Lovegrove and M Ali (London: Gordon and Breach)

[DASG93b] Dasgupta D and McGregor D R 1993 Short term unit-commitment using genetic algorithms *Proc. 5th Int. Conf. on Tools with Artificial Intelligence, TAI'93 (Boston, MA, November 8–11 1993)* (Piscataway, NJ: IEEE) pp 240–7

[DAVE94] Davern J J 1994 An architecture for job scheduling with genetic algorithms *Doctoral Dissertation* University of Central Florida, FL

[DAVI85] Davis L 1985 Job shop scheduling with genetic algorithms. In: [GREF85a] pp 136–40

[DAVI87] Davis L and Coombs S 1987 Genetic algorithms and communication link speed design: theoretical considerations. In: [GREF87a] pp 252–6

[DAVI89] Davis L and Coombs S 1989 Optimizing network link sizes with genetic algorithms *Modelling and Simulation Methodology* ed M S Elzas, T I Ören and B P Zeigler (Amsterdam: North-Holland) pp 317–31

[DAVI90] Davis L 1990 Applying adaptive algorithms to epistatic domains *Proc. 9th Int. Joint Conf. on Artificial Intelligence* vol 1, ed P B Mirchandani and R L Francis (New York: Wiley) pp 162–4

[DAVI91] Davis L 1991 *Handbook of Genetic Algorithms* (New York: Van Nostrand-Reinhold)

[DAVI92] Davidor Y and Ben-Kiki O 1992 The interplay among the genetic algorithm operators: information theory tools used in a holistic way. In: [MÄNN92] pp 75–84

[DAVI93a] Davidor Y, Yamada T and Nakano R 1993 The ECOlogical framework II: improving GA performance at virtually zero cost. In: [FORR93] pp 171–6

[DAVI93b] Davis L, Orvosh D, Cox A and Qiu Y 1993 A genetic algorithm for survivable network design. In: [FORR93] pp 408–15

[DAVI94a] Davidor Y, Schwefel H P and Männer R (ed) 1994 *Parallel Problem Solving from Nature—PPSN III, Int. Conf. on Evolutionary Computation, 3rd Conf. on Parallel Problem Solving from Nature (Jerusalem, October 1994)* (Berlin: Springer)

[DAVI95] Davidson J W and Goulter I C 1995 Evolution program for design of rectilinear branched networks *ASCE J. Comput. Civ. Eng.* **9** 112–21

[DAWI95] Dawid H 1995 Learning by genetic algorithms in evolutionary games *Operations Research Proc. 1994, Selected Papers of the Int. Conf. on Operations Research (Berlin, 30 August–2 September 1994)* ed U Derigs, A Bachem and A Drexl (Berlin: Springer) pp 261–6

[DEGE95] Dege V 1995 Sequencing jobs in the car industry (Skoda), personal communication

[DELA94] Delahaye D, Alliot J M, Schoenauer M and Farges J-L 1994 Genetic algorithms for partitioning air space *Proc. 10th Conf. on Artificial Intelligence for Applications (San Antonio, March 1–4 1994)* (Piscataway, NJ: IEEE) pp 291–7

[DELA95] Delahaye D, Alliot J M, Schoenauer M and Farges J-L 1995 Genetic algorithms for automatic regroupment of air traffic control sectors. In: [MCDO95a]

[DEMM92] Demmel A 1992 Entwurf und Realisierung eines Genetischen Algorithmus zur Standortplanung im Distributionsbereich *Diploma Thesis* Department of Economics, University of Hagen, Germany

[DENB94a] Denby B and Schofield D 1994 Open-pit design and scheduling by use of genetic algorithms *Trans. Mining Metallurgy A: Mining Industry* **103** A21–6

[DENB94b] Denby B and Schofield D 1994 Open-pit design and scheduling by use of genetic algorithm *IEEE Trans. Parallel Distributed Syst.* **5** 113–20

[DENB95] Denby B and Schofield D 1995 The use of genetic algorithms in underground mine scheduling *Australasian Institute of Mining and Metallurgy Publication Series* vol 95 (Adelaide: Austrailain Institute of Mining and Metallurgy) pp 389–94

[DEO95] Deo S 1995 GA for scheduling of jobs for electronics assmbly, personal communication

[DEPP92] Depping J 1992 Kombinierte Touren- und Standortplanung bei der Hausmüllentsorgung mit einem evolutionsstrategischen Ansatz *Diploma Thesis* Department of Computer Science, University of Dortmund, Germany

[DING92] Ding H, El-Keib A A and Smith R E 1992 Optimal clustering of power networks using genetic algorithms *TCGA Report* 92001, University of Illinois at Urbana-Champaign, IL

[DJER95] Djerid L, Portmann M C, Villon P 1995 Performance analysis of previous and new proposed cross-over genetic operators designed for permutation scheduling problems *Proc. Int. Conf. on Industrial Engineering and Production Management (organized by FUCAM) (Mons, Belgium, 4–7 April 1995)* pp 487–97

[DORN92] Dorndorf U and Pesch E 1992 Evolution based learning in a job shop scheduling environment *Research Memorandum* 92-019, University of Limburg, Belgium

[DORN93] Dorndorf U and Pesch E 1993 Genetic algorithms for job shop scheduling *Operations Research Proc., Papers of the 21st Ann. Meeting of DGOR in Cooperation with ÖGOR 1992 (Aachen, 9–11 September 1992)* (Berlin: Springer) pp 243–50

[DORN95] Dorndorf U and Pesch E 1995 Evolution based learning in a job shop scheduling environment *Comput. Operations Res.* **22** 25–40

[DOSI93] Dosi G, Marengo L, Bassanini A and Valente M 1993 Microbehaviours and dynamical systems: economic routines as emergent properties of adaptive learning *Path-Dependent Economics* ed C Antonelli and P A David (Dordrecht: Kluwer)

[DRIE92] Driessche R V and Piessens R 1992 Load balancing with genetic algorithms. In: [MÄNN92] pp 341–50

[DUEC90] Dueck G and Scheuer T 1990 Threshold accepting: a general purpose optimization algorithm appearing superior to simulated annealing *J. Comput. Phys.* **90** 161–75

[DWOR96] Dworman G, Kimbrough S O and Laing J D 1996 Bargaining by artificial agents in two coalition games: a study in genetic programming for electronic commerce. In: [KOZA96a]

[DZUB94] Dzubera J and Whitley D 1994 Advanced correlation analysis of operators for the traveling salesman problem. In: [DAVI94a] pp 68–77

[EAST93] Easton F F and Mansour N 1993 A distributed genetic algorithm for employee staffing and scheduling problems. In: [FORR93] pp 360–7

[EBEL90] Ebeling W 1990 Applications of evolutionary strategies *Syst. Analysis Modeling Simulation* **7** 3–16

[EDDE95] Eddelbüttel D 1995 Optimized stock investment, personal communication

[EGLI94] Eglit J T 1994 Trend prediction in financial time series. In: [KOZA94b] pp 31–40

[EIBE94a] Eiben A E, Raué P E and Ruttkay Z 1994 Genetic algorithms with multi-parent recombination. In: [DAVI94a] pp 78–87

[EIBE94b] Eiben A E and Hartsuiker L 1994 Resource allocation by genetic algorithms *Proc. EUFIT'94 (Aachen)* (Aachen: ELITE Foundation) pp 1679–82

[EIBE96] Eiben A E, Euverman T, Kowalczyk W and Peelen E, Slisser F and Wesseling J 1996 Genetic algorithms and neural networks vs statistical techniques: a case study in marketing *Research Paper* University of Utrecht, The Netherlands

[ERBE95] Erben W 1995 Timetabling using genetic algorithms. In: [PEAR95] pp 30–2

[ERGU95] Ergul A 1995 Timetabling of exams and classes *Timetabling Problems List (E-Mail List)* 31.1.1995

[ESBE95] Esbensen H 1995 Finding (near-) optimal Steiner trees in large graphs. In: [ESHE95] pp 485–91

[ESHE91] Eshelman L J 1991 The CHC adaptive search algorithm: how to have safe search when engaging in nontraditional genetic recombination. In: [RAWL91] pp 265–83

[ESHE95] Eshelman L J (ed) 1995 *Proc. 6th Int. Conf. on Genetic Algorithms* (San Fransico, CA: Morgan Kaufmann)

[ETTL95] Ettl M and Schwehm M 1995 Determining the optimal network partition and kanban allocation in JIT production lines. In: [BIET95a] pp 139–52

[FALC93] De Falco I, Del Balio R and Tarantino E 1993 Testing parallel evolution strategies on the quadratic assignment problem *Proc. Int. Conf. on Systems, Man and Cybernetics (Le Touquet, France, 17–20 October 1993)* vol 5 (Piscataway, NJ: IEEE) pp 254–9

[FALC95] De Falco I, Del Balio R and Tarantino E 1995 Comparing parallel tabu search and parallel genetic algorithms on the task allocation problem. In: [PEAR95] pp 380–3

[FALK80] von Falkenhausen K 1980 Optimierung regionaler Entsorgungssysteme mit der Evolutionsstrategie *Proceedings in Operations Research* vol 9, ed J Schwarze *et al* (Würzburg: Physica) pp 46–51

[FALK91a] Falkenauer E and Bouffouix S 1991 A genetic algorithm for job shop *Proc. 1991 IEEE Int. Conf. on Robotics and Automation (Sacramento, CA, April 1991)* (Piscataway, NJ: IEEE) pp 824–9

[FALK91b] Falkenauer E 1991 A genetic algorithm for grouping *Proc. 5th Int. Symp. on Applied Stochastic Models and Data Analysis (Granada, 23–6 April 1991)* (Singapore: World Scientific) pp 198–206

[FALK92] Falkenauer E and Delchambre A 1992 A genetic algorithm for bin packing and line balancing *Proc. 1992 IEEE Int. Conf. on Robotics and Automation (Nice, May 1992)* (Piscataway, NJ: IEEE) pp 1186–92

[FALK94a] Falkenauer E 1994 A hybrid grouping genetic algorithm for bin packing *Research Paper* (Brussels: CRIF Industrial Management and Automation)

[FALK94b] Falkenauer E 1994 A new representation and operators for genetic algorithms applied to grouping problems *Evolutionary Computat.* **2** 123–44

[FALK95] Falkenauer E 1995 Tapping the full power of genetic algorithm through suitable representation and local optimization: application to bin packing. In: [BIET95a] pp 167–82

[FAND92] Fandel G, Gulledge T and Jones A (ed) 1992 *New Directions for Operations Research in Manufacturing, Proc. Joint US/German Conf. (Gaithersburg, MD, July 30–31 1991)* (Berlin: Springer)

[FAND93] Fandel G, Gulledge T and Jones A (ed) 1993 *Operations Research in Production Planning and Control, Proc. Joint German/US-Conf. 1992* (Berlin: Springer)

[FANG92] Fang H L 1992 Investigating GAs for scheduling *MSc Dissertation* Department of Artificial Intelligence, University of Edinburgh, Edinburgh, UK

[FANG93] Fang H L, Ross, P and Corne D 1993 A promising genetic algorithm approach to job-shop scheduling, rescheduling, and open-shop scheduling problems. In: [FORR93] pp 375–82

[FICH95] Fichera S, Grasso V, Lombardo A and Lovalvo E 1995 Genetic algorithms efficiency in flow-shop scheduling *Applications Artificial Intelligence Eng.* **10** 261–70

[FILI92] Filipic B 1992 Enhancing genetic search to schedule a production unit *ECAI 92, Proc. 10th Eur. Conf. on Artificial Intelligence (Vienna, 3–7 August 1992)* ed B Neumann B (New York: Wiley) pp 603–7

[FILI93] Filipic B 1993 Enhancing genetic search to schedule a production unit *Scheduling of Production Processes* ed J Dorn and K A Froeschl (Chichester: Ellis Horwood) pp 61–9

[FILI94] Filipic B and Srdoc A 1994 Task scheduling and resource management in ship repair using a genetic algorithm *Proc. 8th Int. Conf. on Computer Applications in Shipbuilding* vol 2, ed J Brodda and K Johansson (Bremen: IFIP) pp 15.17–15.28

[FILI95] Filipic B 1995 A genetic algorithm applied to resource management in production systems. In: [BIET95a] pp 101–11

[FLEU94] Fleurent C and Ferland J A 1994 Genetic hybrids for the quadratic assigment problem *DIMACS Series in Discrete Mathematics and Theoretical Computer Science* vol 16, ed P M Pardalos and H Wolkowicz (Providence, RI: American Mathematical Society) pp 173–88

[FLEU95] Fleurent C and Ferland J A 1995 Object-oriented implementation of heuristic search methods for graph coloring, maximum clique and satisfiability *Second DIMACS Challenge (DIMACS Series in Discrete Mathematics and Theoretical Computer Science)* ed M A Trick and D S Johnson, Special Issue (Providence, RI: American Mathematical Society) to appear

[FLEU96] Fleurent C and Ferland J A 1995 Genetic and hybrid algorithms for graph coloring *Metaheuristics in Combinatorial Optimization* ed G Laporte, I H Osman and P L Hammer *Ann. Operations Res.* Special Issue **63**

[FOGA88] Fogarty T C 1988 Rule-based optimisation of combustion in multiple burner furnaces and boiler plants *Eng. Applications Artificial Intelligence* **1** 203–9

[FOGA89] Fogarty T C 1989 Learning new rules and adapting old ones with the genetic algorithm *Artificial Intelligence in Manufacturing, Proc. 4th Int. Conf. on Applications of Artificial Intelligence in Engineering* ed G Rzevski G (Berlin: Springer) pp 275–90

[FOGA91] Fogarty T C 1991 Credit scoring and control applications of the genetic algorithm *Parallel Problem Solving from Nature—Applications in Statistics and Economics, PASE Workshop Proc. (Zürich 1991)* (Zürich: ETH Zürich) pp 147–8

[FOGA92] Fogarty T C 1992 Developing rule-based systems for credit card applications from data with the genetic algorithm *IMA J. Math. Appl. Business Ind.* **4** 53–9

[FOGA93] Fogarty T C and Cui J 1993 Optimization of a 532-city symmetric travelling salesman problem with a parallel genetic algorithm *Proc. IMA Conf. on Parallel Computing* ed A E Fincham and B Ford (Oxford: Claredon) pp 295–304

[FOGA94a] Fogarty T C (ed) 1994 *Evolutionary Computing, AISB Workshop (Leeds, UK, 11–13 April 1994), Selected Papers (Lecture Notes in Computer Science 865)* (Berlin: Springer)

[FOGA94b] Fogarty T C and Ireson N S 1994 Evolving Bayesian classifiers for credit control—a comparison with other machine learning methods *IMA J. Math. Appl. Business Ind.* **5** 63–75

[FOGA95a] Fogarty T C 1995 *Proposal for a Network of Excellence in Evolutionary Computing (EvoNet)* pt 2 (Bristol, UK: University of the West of England)

[FOGA95b] Fogarty T C 1995 Scheduling at Daimler-Benz and Rolls Royce, personal communication
[FOGA95c] Fogarty T C (ed) 1995 *Evolutionary Computing, AISB Workshop, Selected Papers (Lecture Notes in Computer Science 993)* (Berlin: Springer)
[FOGA95d] Fogarty T C, Vavak F and Cheng P 1995 Use of the genetic algorithm for load balancing of sugar beet presses. In: [ESHE95] pp 617–24
[FOGE66] Fogel L J, Owens A J and Walsh M J 1966 *Artificial Intelligence Through Simulated Evolution* (New York: Wiley)
[FOGE88] Fogel D B 1988 An evolutionary approach to the traveling salesman problem *Biol. Cybern.* **60** 139–44
[FOGE90] Fogel D B 1990 A parallel processing approach to a multiple traveling salesman problem using evolutionary programming *Proc. 4th Ann. Symp. on Parallel Processing* ed L H Canter (Fullerton, CA: IEEE Orange County Computer Society) pp 318–26
[FOGE91] Fogel D B 1991 The evolution of intelligent decision making in gaming *Cybern. Syst.* **22** 223–36
[FOGE92a] Fogel D B 1992 Evolving artificial intelligence *Doctoral Dissertation* University of California at San Diego, CA
[FOGE92b] Fogel D B and Atmar W (ed) 1992 *Proc. 1st Ann. Conf. on Evolutionary Programming (La Jolla, CA, February 21–22, 1992)* (San Diego, CA: Evolutionary Programming Society)
[FOGE93a] Fogel D B 1993 Evolving behaviors in the iterated prisoner's dilemma *Evolutionary Computat.* **1** 77–97
[FOGE93b] Fogel D B and Atmar W (ed) 1993 *Proc. 2nd Ann. Conf. on Evolutionary Programming* (San Diego, CA: Evolutionary Programming Society)
[FOGE93c] Fogel D B 1993 Applying evolutionary programming to selected travelling salesman problems *Cybern. Syst.* **24** 27–36
[FOGE93d] Fogel D B 1993 Empirical estimation of the computation required to reach approximate solutions to the travelling salesman problem using evolutionary programming. In: [FOGE93b] pp 56–61
[FOGE94] Fogel D B 1994 Evolving continuous behaviors in the iterated prisoner's dilemma. In: [SEBA94] pp 119–30
[FOGE95a] Fogel D B 1995 Evolutionary computation *Toward a New Philosophy of Machine Intelligence* (Piscataway, NJ: IEEE)
[FOGE95b] Fogel D B 1995 On the relationship between the duration of an encounter and evolution of cooperation in the iterated prisoner's dilemma *Evolutionary Computat.* **3** 349–63
[FOGE96] Fogel D B 1996 1. Inventory control project for a major engine manufacturer in the USA; 2. Scheduling delivery fuel trucks for a major oil company in the USA; 3. Online reassignment of computer tasks across a suite of heterogeneous computers; 4. Evolved neural network predictor to handle pension money based on investing in commodities; 5. Factory scheduling for a finishing plant for a major US clothing company; 6. Optimizing the cutting of fabric; 7. Mission planning software for the US Government and for military applications; personal communication with D B Fogel
[FORR93] Forrest S (ed) 1993 *Proc. 5th Int. Conf. on Genetic Algorithms* (San Mateo, CA: Morgan Kaufmann)
[FORS95] Forsyth P 1995 Timetabling with a predator–prey model *Timetabling Problems List (E- Mail List)* 21.8.1995
[FOUR85] Fourman M P 1985 Compaction of symbolic layout using genetic algorithms. In: [GREF85a] pp 141–53
[FOX91] Fox B R and McMahon M B 1991 Genetic operators for sequencing problems. In: [RAWL91] pp 284–300
[FUCH83] Fuchs F and Maier H A 1983 Optimierung des Lastflusses in elektrischen Energieversorgungsnetzen mittels Zufallszahlen *Archiv Elektrotechnik* **66** 85–94
[FUJI87] Fujiki C and Dickinson J 1987 Using the genetic algorithm to generate LISP source code to solve the prisoner's dilemma. In: [GREF87a] pp 236–45
[FUJI95] Fujimoto H, Yasuda K, Tanigawa Y and Iwahashi K 1995 Applications of genetic algorithm and simulation to dispatching rule-based FMS scheduling *Proc. IEEE Int. Conf. On Robotics and Automation* (Piscataway, NJ: IEEE) pp 190–5
[FUKU93] Fukuda T, Mori K and Tsukiyama M 1993 Immune networks using genetic algorithm for adaptive production scheduling *Automatic Control—World Congress 1993 (Sydney, 18–23 July 1993)* vol 4, ed G C Goodwin and R J Evans (Oxford: Pergamon) pp 353–6
[FUKU95] Fukushima M 1995 Generating class timetable *Timetabling Problems List (E-Mail List)* 24.2.1995
[FULK93a] Fulkerson W 1993 Scheduling in a foundry, personal communication
[FULK93b] Fulkerson W 1993 Scheduling assembly lines, personal communication
[GABB91] Gabbert P S *et al* 1991 A system for learning routes and schedules with genetic algorithms. In: [BELE91] pp 430–6
[GAMM91] Gammack J G, Fogarty T C, Battle S A and Miles R G 1991 Management decision support from large databases: the IDIOMS project *Proc. AMSE Int. Conf. on Signals and Systems (Warsaw, 1991)* vol 1, pp 213–19
[GARC96] Garces-Perez J, Schoenfeld D A and Wainwright R L 1996 Solving facility layout problems using genetic programming. In: [KOZA96a]
[GARI91] de Garis H 1991 Genetic programming: building artificial nervous systems with genetically programmed neural network modules *Neural and Intelligent Systems Integration* ed B Soucek and the IRIS Group (New York: Wiley) pp 207–34

[GEHR94] Gehring H and Schütz G 1994 Zwei Genetische Algorithmen zur Lösung des Bandabgleichproblems *Operations Research. Reflexionen aus Theorie und Praxis* ed B Werners and R Gabriel (Berlin: Springer)

[GEN94a] Gen M, Tsujimura Y and Kubota E 1994 Solving job-scheduling problem with fuzzy processing time using genetic algorithm *Proc. 2nd Eur. Congress on Intelligent Techniques and Soft Computing (EUFIT'94) (Aachen, 20–23 September 1994)* vol 3 (Aachen: ELITE Foundation) pp 1540–7

[GEN94b] Gen M, Tsujimura Y and Kubota E 1994 Solving job-shop scheduling problems by genetic algorithm *Proc. 1994 IEEE Int. Conf. on Systems, Man and Cybernetics—Humans, Information and Technology* vols 1–3 (Piscataway, NJ: IEEE) pp 1577–82

[GEN94c] Gen M, Ida K, Kono E and Li Y 1994 Solving bicriteria solid transportation problem by genetic algorithm *Proc. 16th Int. Conf. on Computing and Industrial Engineering (Ashikaga, Japan, 1994)* ed M Gen and G Yamazaki (Oxford: Pergamon) pp 572–5

[GEN95] Gen M, Ida K and Li Y 1995 Solving bicriteria solid transportation problem with fuzzy numbers by genetic algorithm *Proc. 17th Int. Conf. on Computing and Industrial Engineering (Phoenix, AZ)* ed C P Koelling (Oxford: Pergamon) p 3

[GERD94a] Gerdes I S 1994 Application of genetic algorithms to the problem of free-routing of aircraft *Proc. ICEC'94 (Orlando, FL, 1994)* vol II (Piscataway, NJ: IEEE) pp 536–41

[GERD94b] Gerdes I 1994 Construction of conflict-free routes for aircraft in case of free-routing with genetic algorithms. In: [KUNZ94] pp 97–8

[GERD94c] Gerdes I 1994 Construction of conflict-free routes for aircraft in case of free-routing with genetic algorithms. In: [HOPF94] pp 143–9

[GERD95] Gerdes I 1995 Application of genetic algorithms for solving problems related to free routing for aircraft. In: [BIET95a] pp 328–40

[GERR91] Gerrits M and Hogeweg P 1991 Redundant coding of an NP-complete problem allows effective genetic algorithm search. In: [SCHW91] pp 70–4

[GERR93] Gerrets M, Walker R and Haasdijk E 1993 Time series prediction of financial data, personal communication

[GEUR93] Geurts M 1993 Job shop scheduling, personal communication

[GOFF94] Goffe W L, Ferrier G D and Rogers J 1994 Global optimization of statistical functions with simulated annealing *J. Econometrics* **60** 65–99

[GOLD85] Goldberg D E and Lingle R Jr 1985 Alleles, loci, and the traveling salesman problem. In: [GREF85a] pp 154–9

[GOLD87] Goldberg D E and Smith R E 1987 Nonstationary function optimization using genetic algorithms with dominance and diploidy. In: [GREF87a] pp 59–68

[GOLD89] Goldberg D E 1989 *Genetic Algorithms in Search, Optimization, and Machine Learning* (Reading, MA: Addison-Wesley)

[GONZ94] Gonzalez C and Wainwright R L 1994 Dynamic scheduling of computer tasks using genetic algorithms *Proc. 1st IEEE Conf. on Evolutionary Computing (Orlando, FL, June 27–29 1994)* (Piscataway, NJ: IEEE) pp 829–33

[GOON94] Goonatilake S and Feldman K 1994 Genetic rule induction for financial decision making *Genetic Algorithms in Optimisation, Simulation and Modelling (Frontiers in Artificial Intelligence and Applications 23)* ed J Stender, E Hillebrand and J Kingdon (Amsterdam: IOS)

[GOON95a] Goonatilake S and Treleaven P 1995 *Intelligent Systems for Finance and Business* (New York: Wiley)

[GOON95b] Goonatilake S, Campbell J A and Ahmad N 1995 Genetic-fuzzy systems for financial decision making *Advances in Fuzzy Logic, Neural Networks and Genetic Algorithms (Lecture Notes in Computer Science 1011)* ed T Furuhashi (Berlin: Springer) pp 202–23

[GORD93] Gordon V S and Whitley D 1993 Serial and parallel genetic algorithms as function optimizers. In: [FORR93] pp 177–83

[GORG89] Gorges-Schleuter M 1989 ASPARAGOS: an asynchronous parallel genetic optimization strategy. In: [SCHA89] pp 422–7

[GORG91a] Gorges-Schleuter M 1991 Explicit parallelism of genetic algorithms through population structures. In: [SCHW91] pp 150–9

[GORG91b] Gorges-Schleuter M 1991 ASPARAGOS: a parallel genetic algorithm and population genetics. In: [BECK91] pp 407–18

[GORG91c] Gorges-Schleuter M 1991 Genetic algorithms and population structures. A massively parallel algorithm *Doctoral Dissertation* Department of Computer Science, University of Dortmund, Germany

[GREE87] Greene D P and Smith S F 1987 A genetic system for learning models of consumer choice. In: [GREF87a] pp 217–23

[GREE94] Greenwood G 1994 Preventative maintenance tasks *Genetic Algorithms Digest (E-mail List)* 8 (15) 10.5.1994

[GREF85a] Grefenstette J J (ed) 1985 *Proc. Int. Conf. on Genetic Algorithms and Their Applications (Pittsburgh, PA, July 24–26)* (Hillsdale, NJ: Erlbaum)

[GREF85b] Grefenstette J, Gopal R, Rosmaita B and Gucht D V 1985 Genetic algorithms for the traveling salesman problem. In: [GREF85a] pp 160–8

[GREF85c] Grefenstette J J and Fitzpatrick J M 1985 Genetic search with approximate function evaluations. In: [GREF85a] pp 112–20

[GREF87a] Grefenstette J J (ed) 1987 Genetic algorithms and their applications *Proc. 2nd Int. Conf. on Genetic Algorithms (MIT, Cambridge, MA, July 28–31)* (Hillsdale, NJ: Erlbaum)

[GREF87b] Grefenstette J J 1987 Incorporating problem specific knowledge into genetic algorithms *Genetic Algorithms and Simulated Annealing* ed L Davis (San Mateo, CA: Morgan Kaufmann) pp 42–59

[GROO91] de Groot C, Würtz D and Hoffmann K H 1991 Optimizing complex problems by Nature's algorithms: simulated annealing und evolution strategy—a comparative study. In: [SCHW91] pp 445–54

[GRÜN95] Grünert T 1995 Lotsizing and scheduling *Genetic Algorithms in Production Scheduling List (E-Mail List)* 26.1.1995

[GUPT92] Gupta Y P, Sundaram C, Gupta M C and Kumar A 1992 Minimizing total intercell and intracell moves in cellular manufacturing: a genetic algorithm approach *Proc. Ann. Meeting of the Decision Science Institute (San Francisco, November 22–24)* (Philadelphia, PA: American Institute for Decision Sciences) pp 1523–5

[GUPT93a] Gupta M C, Gupta Y P and Kumar A 1993 Minimizing flow time variance in a single machine system using genetic algorithm. *Eur. J. Operations Res.* **70** 289–303

[GUPT93b] Gupta M C, Gupta Y P and Kumar A 1993 Genetic algorithms application in a machine scheduling problem *Proc. ACM Computer Science Conf.* (New York: ACM) pp 372–77

[GÜVE95] Güvenir H A 1995 A genetic algorithm for multicriteria inventory classification. In: [PEAR95] pp 6–9

[HADI95a] Hadinoto D 1995 Single machine scheduling problems and permutation flow shop problems. *Genetic Algorithms in Production Scheduling List (E-mail List)* 15.3.1995

[HADI95b] Hadinoto D 1995 Combinations of branch&bound techniques with genetic algorithms for scheduling problems *Genetic Algorithms in Production Scheduling List (E-mail List)* 4.9.1995

[HADJ92] Ben Hadj-Alouane A and Bean J C 1992 A genetic algorithm for the multiple-choice integer program *Technical Report* 92-50, Department of Industrial and Operations Research, University of Michigan, Ann Arbor, MI

[HAMM94] Hammel U and Bäck T 1994 Evolution strategies on noisy functions—how to improve convergence properties. In: [DAVI94a] pp 159–68

[HAMP81] Hampel C 1981 Ein Vergleich von Optimierungsverfahren für die zeitdiskrete Simulation *Doctoral Dissertation* Technical University of Berlin

[HANN89] Hannan M T and Freeman J 1989 *Organizational Ecology* (Cambridge, MA: Harvard University Press)

[HANS95] Hansen S 1995 PROFIT, Software für die Prognose, Planung und Steuerung der Produktion. In: [BIET95b] pp 175–80

[HAO95] Hao J K 1995 A clausal genetic representation and its related evolutionary procedures for satisfiability problems. In: [PEAR95] pp 289–92

[HART95] Hartley S J 1995 GA to solve the Boolean satisfiability problem *Genetic Algorithms Digest (E-mail List)* 9 (36) 22.6.1995

[HART96] Hartmann S 1996 Projektplanung bei knappen Mitteln *PhD Project* Faculty of Social and Business Sciences, Christian-Albrechts-University, Kiel

[HEIS94a] Heistermann J 1994 Genetic algorithms at Siemens. In: [KUNZ94] pp 100–1

[HEIS94b] Heistermann J 1994 Genetic algorithms at Siemens. In: [HOPF94] pp 150–9

[HELB93] Helber S 1993 Lösung dynamischer mehrstufiger Mehrprodukt-Losgrössenprobleme unter Kapazitätsrestriktionen durch lokale Suchverfahren *Operations Research Proc., Papers of the 21st Ann. Meeting of DGOR in Cooperation with ÖGOR 1992 (Aachen, 9–11 September 1992)* (Berlin: Springer) pp 133

[HENS86] Hense A V 1986 Adaptionsstrategien zur Lösung schwieriger Optimierungsaufgaben *Diploma Thesis* Department of Computer Science, University of Dortmund, Germany

[HERD88] Herdy M 1988 Anwendung der Evolutionsstrategie auf das Travelling Salesman Problem *Thesis* Department of Bionics, Technical University of Berlin

[HERD91] Herdy M 1991 Application of the Evolutionsstrategie to discrete optimization problems. In: [SCHW91] pp 188–92

[HESS89] Hesser J, Männer R and Stucky O 1989 Optimization of steiner trees using genetic algorithms. In: [SCHA89] pp 231–6

[HESS91] Hesser J, Männer R and Stucky O 1991 On Steiner trees and genetic algorithms. In: [BECK91] pp 509–25

[HEUS70] Heusener G 1970 Optimierung natrium-gekühlter schneller Brutreaktoren mit Methoden der nicht-linearen Programmierung *Doctoral Dissertation* University of Karlsruhe, Germany

[HILL87] Hilliard M R, Liepins G E, Palmer M, Morrow M and Richardson J 1987 A classifier-based system for discovering scheduling heuristics. In: [GREF87a] pp 231–5

[HILL88] Hillard M R, Liepins G E and Palmer M R 1988 Machine learning applications to job shop scheduling *Proc. 1st Int. Conf. on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA/AIE-88) (University of Tennessee, Tullahoma, June 1–3 1988)* ed M Ali (New York: ACM) pp 728–37

[HILL89a] Hilliard M R, Liepins G, Rangarajan G and Palmer M 1989 Learning decision rules for scheduling problems: a classifier hybrid approach *Proc. 6th Int. Workshop on Machine Learning* ed A M Segre (San Mateo, CA: Morgan Kaufmann) pp 188–190

[HILL89b] Hilliard M R, Liepins G E and Palmer M and Rangarajan G 1989 The computer as a partner in algorithmic design: automated discovery of parameters for a multi-objective scheduling heuristic *Impacts of Recent Computer Advances on Operations Research* ed R Sharda, B Golden, E Wasil, O Balci and W Steward (Amsterdam: North-Holland) pp 321–31

[HILL90] Hilliard M R, Liepins G E and Palmer M 1990 Discovering and refining algorithms through machine learning *Operations Research and Artificial Intelligence: The Integration of Problem-Solving Strategies* ed D E Brown and C C White (Dordrecht: Kluwer) pp 59–78

[HILL94] Hillebrand E and Stender J (ed) 1994 *Many-Agent Simulation and Artificial Life (Frontiers in Artificial Intelligence and Applications 25)* (Amsterdam: IOS)

[HINT94] Hinterding R 1994 A new way of solving the bin packing problem using GAs *Genetic Algorithms Digest (E-mail List)* 8 (15) 10.5.1994

[HIRA95] Hirayama K 1995 Slab design problem in steel production (involves sequencing order plates, deciding slab sizes and total number of slabs) *Genetic Algorithms Digest (E-mail List)* 9 (37) 28.6.1995

[HIRZ92] Hirzer J 1992 Lösung des Standard-Tourenplanungsproblems mit einem Genetischen Algorithmus *Diploma Thesis* Department of Economics, University of Hagen, Germany

[HO96] Ho T H 1996 Finite automata play repeated prisoner's dilemma with information processing costs *J. Econ. Dynamics Control* **20** 173–207

[HOEH96] Hoehfeld M 1996 Load management for power stations at Siemens, Germany, talk presented at: 'Evolutionary Algorithms and Their Applications' Seminar, Dagstuhl, Germany, March 1996

[HOEL96a] Hoelting C J, Schoenefeld D A and Wainwright R L 1996 A genetic algorithm for the minimum broadcast time problem using a global precedence vector *Proc. 1996 ACM/SIGAPP Symp. on Applied Computing* (New York: ACM) pp 258–62

[HOEL96b] Hoelting C J, Schoenefeld D A and Wainwright R L 1996 Finding investigator tours in telecommunication networks using genetic algorithms *Proc. 1996 ACM/SIGAPP Symp. on Applied Computing* (New York: ACM) pp 82–7

[HOFF91a] Hoffmeister F and Bäck T 1991 Genetic algorithms and evolution strategies: similarities and differences. In: [SCHW91] pp 455–69

[HOFF91b] Hoffmeister F 1991 Scalable parallelism by evolutionary algorithms *Parallel Computing and Mathematical Optimization* ed M Grauer and D B Pressmar (Berlin: Springer) pp 177–98

[HÖHN95] Höhn C and Reeves C 1995 Incorporating neighbourhood search operators into genetic algorithms. In: [PEAR95] pp 214–7

[HOLL86] Holland J H, Holyoak K J, Nisbett R E and Thagard P R 1986 *Induction: Processes of Inference, Learning, and Discovery* (Cambridge, MA: MIT Press)

[HOLL92a] Holland J H 1992 *Adaptation in Natural and Artificial Systems* (Cambridge, MA: MIT Press) 2nd edn

[HOLL92b] Holland J H 1992 Genetische Algorithmen *Spektrum der Wissenschaft* **9** 44–51

[HOLM95] Holmes J, Routen T W and Czarnecki C A 1995 Heterogeneous co-evolving parasites. In: [PEAR95] pp 156–9

[HOLS93] Holsapple C W, Jacob V S, Pakath R and Zaveri J S 1993 A genetic-based hybrid scheduler for generating static schedules in flexible manufacturing contexts *IEEE Trans. Syst., Man Cybern.* **23** 953–72

[HOMA93] Homaifar A, Guan S and Liepins G E 1993 A new approach on the traveling salesman problem by genetic algorithms. In: [FORR93] pp 460–6

[HÖNE90] Hönerloh A 1990 Der Einsatz von Evolutionsstrategien zur Entscheidungsvorbereitung in der betrieblichen Ablaufplanung *Diploma Thesis* Department of Economics, University of Göttingen

[HOPF94] Hopf J (ed) 1994 Genetic algorithms within the framework of evolutionary computation *Proc. KI-94 Workshop, Technical Report* MPI-I-94-241, Max-Planck-Institut für Informatik, Saarbrücken

[HOU90] Hou E S H, Hong R and Ansari N 1990 Efficient multiprocessor scheduling based on genetic algorithms *Power Electronics Emerging Technologies, IECON Proc.* vol 2 (New York: IEEE) pp 1239–43

[HOU92] Hou E S H, Ren H and Ansari N 1992 Efficient multiprocessor scheduling based on genetic algorithms. In: [SOUC92] pp 339–52

[HOU94] Hou E S H, Ansari N and Ren H A 1994 A genetic algorithm for multiprocessor scheduling. *IEEE Trans. Parallel Distrib. Syst.* **5** 113–120

[HUGH90] Hughes M 1990 Improving products and process—Nature's way *Ind. Management Data Syst.* **6** 22–5

[HULI91] Hulin M 1991 Circuit partitioning with genetic algorithms using a coding scheme to preserve the structure of a circuit. In: [SCHW91] pp 75–9

[HULI92] Hulin M 1992 Evolution strategies for circuit partitioning. In: [SOUC92] pp 413–37

[HÜLS94] Hülsemann M 1994 Schwellkettenoptimierung bei Wasserkraftwerken, paper presented at: Int. Conf. on Operations Research, TU Berlin, 30 August–2 September

[HUNT91] Huntley C L and Brown D E 1991 A parallel heuristic for quadratic assignment problems *Comput. Operations Res.* **18** 275–89

[HURL94] Hurley S M L and Stephens N M 1994 Applying genetic algorithms to problems in marketing *Proc. 23rd EMAC Conf., Marketing: Its Dynamics and Challenges* ed J Bloemer, J Lemmink and H Kasper (Maastricht: European Marketing Academy)

[HURL95] Hurley S, Moutinho L and Stephens N M 1995 Solving marketing optimization problems using genetic algorithms *Research Paper* University of Cardiff, UK

[HUSB91a] Husbands P and Mill F 1991 Simulated co-evolution as the mechanism for emergent planning and scheduling. In: [BELE91] pp 264–70

[HUSB91b] Husbands P, Mill F and Warrington S 1991 Genetic algorithms, production plan optimisation and scheduling. In: [SCHW91] pp 80–4

[HUSB92] Husbands P 1992 Genetic algorithms in optimisation and adaption *Advances in Parallel Algorithms* ed L Kronsjo and D Shumsheruddin (Oxford: Blackwell Scientific) pp 227–77

[HUSB93] Husbands P 1993 An ecosystem model for integrated production planning *Int. J. Computer Integrated Manufacturing* **6** 74–86

[HUSB94] Husbands P 1994 Distributed coevolutionary genetic algorithms for multi-criteria and multi-constraint optimisation. In: [FOGA94a] pp 150–65

[IDA95] Ida K, Gen M and Li Y 1995 Solving multicriteria solid transportation problem with fuzzy numbers by genetic algorithm *Proc. 3rd Eur. Conf. on Intelligent Techniques and Soft Computing (EUFIT'95) (Aachen)* (Aachen: ELITE Foundation) pp 434–441

[IGNI91] Ignizio J P 1991 *Introduction to Expert Systems* (New York: McGraw-Hill)

[IGNI93] Ignizio J P 1993 A hybrid expert system for process scheduling, paper presented at: Workshop on Genetic Algorithms in the Petroleum Industry, Aberdeen, 1993

[INAY94] Inayoshi H and Manderick B 1994 The weighted graph bi-partitioning problem: a look at GA. In: [DAVI94a] pp 617–25

[INGB92] Ingber L and Rosen B 1992 Genetic algorithms and very fast simulated annealing—a comparison *Math. Comput. Model.* **16** 87–100

[IRES94] Ireson N S 1994 Evolving a classifier for the TSB loan application data *Research Paper* University of the West of England, Bristol, UK

[IRES95] Ireson N S and Fogarty T C 1995 Evolving decision support models for credit control. In: [BIET95a] pp 264–76

[ISHI94] Ishibuchi H, Yamamoto N, Murata T and Tanaka H 1994 Genetic algorithms and neighborhood search algorithms for fuzzy flowshop scheduling problems *Fuzzy Sets Syst.* **67** 81–100

[JACK95] Jackson K and Weare R 1995 Generating exam timetables, personal communication

[JACU95] Jacucci G, Foy M and Uhrik C 1995 Genetic algorithms (GAs) in the role of intelligent regional adaptation agents for agricultural decision support systems. In: [PEAR95] pp 428–31

[JAGA94] Jagabandhu S and Chandrasekharan R 1994 Genetic algorithm for family and job scheduling in a flowline-based manufacturing cell *Comput. Ind. Eng.* **27** 469–72

[JAKO94a] Jakobs S 1994 On genetic algorithms for the packing of polygons. In: [KUNZ94] pp 81–2

[JAKO94b] Jakobs S 1994 On genetic algorithms for the packing of polygons. In: [HOPF94] pp 84–5

[JOG87] Jog P and Gucht D V 1987 Parallelisation of probabilistic sequential search algorithms. In: [GREF87a] pp 170–6

[JOG91] Jog P, Suh J Y and Van Gucht D 1991 Parallel genetic algorithms applied to the travelling salesman problem *SIAM J. Optimization* **1** 515–29

[JOHN90] Johnson D S 1990 Local optimization and the traveling salesman problem *Proc. 17th Colloq. on Automata, Languages, and Programming* (Berlin: Springer) pp 446–61

[JONE91] Jones D R and Beltramo M A 1991 Solving partitioning problems with genetic algorithms. In: [BELE91] pp 442–9

[JONE93] Jones A and Rabelo L 1993 Integrating neural nets, simulation, and genetic algorithms for real-time scheduling. In: [FAND93] pp 550–66

[JONG89] De Jong K A and Spears W M 1989 Using genetic algorithms to solve NP-complete problems. In: [SCHA89] pp 124–32

[JONG93] De Jong K A 1993 Genetic algorithms are NOT function optimizers. In: [WHIT93] pp 5–17

[JORD94] Jordan C 1994 A two-phase genetic algorithm to solve variants of the batch sequencing problem *Technical Report* 363, Department of Economics, University of Kiel

[JŐZE95] Józefowska J, Rózycki R and Weglarz J 1995 A genetic algorithm for some discrete–continuous scheduling problems. In: [PEAR95] pp 273–6

[JULI92] Juliff K 1992 Using a multi chromosome genetic algorithm to pack a truck *Technical Report* RMIT CS TR 92-2, Version 1.0, August 1992, Department of Computer Science, Royal Melbourne Institute of Technology, Australia

[JULI93] Juliff K 1993 A multi-chromosome genetic algorithm for pallet loading. In: [FORR93] pp 467–73

[JULS93] Julstrom B A 1993 A genetic algorithm for the rectilinear Steiner problem. In: [FORR93] pp 474–80
[JULS95] Julstrom B A 1995 What have you done for me lately? Adapting operator probabilities in a steady-state genetic algorithm. In: [ESHE95] pp 81–7
[JULS96] Julstrom B A 1996 Contest length, noise, and reciprocal altruism in the population of a genetic algorithm for the iterated prisoner's dilemma. In: [KOZA96b]
[JUNG95] Junginger W 1995 Course scheduling by genetic algorithms. In: [BIET95a] pp 357–70
[KADA90a] Kadaba N 1990 XROUTE: a knowledge-based routing system using neural networks and genetic algorithms *Doctoral Dissertation* North Dakota State University, Fargo, ND
[KADA90b] Kadaba N and Nygard K E 1990 Improving the performance of genetic algorithms in automated discovery of parameters *Proc. 7th Int. Conf. on Machine Learning* ed B W Porter and R J Mooney (San Mateo, CA: Morgan Kaufmann) 140–8
[KADA91] Kadaba N, Mygard K E and Juell P L 1991 Integration of adaptive machine learning and knowledge-based systems for routing and scheduling applications *Expert Syst. Applications* **2** 15–27
[KADO95] Kado K, Ross P and Corne D 1995 A study of genetic algorithm hybrids for facility layout problems. In: [ESHE95] pp 498–505
[KAHN95] Kahng A B and Moon B R 1995 Toward more powerful recombinations. In: [ESHE95] pp 96–103
[KANE91] Kanet J J and Sridharan V 1991 PROGENITOR: a genetic algorithm for production scheduling *Wirtschaftsinformatik* **33** 332–6
[KAPS93] Kapsalis A, Rayward-Smith V J and Smith G D 1993 Solving the graphical Steiner tree problem using genetic algorithms *J. Operations Res. Soc.* **44** 397–406
[KAPS94] Kapsalis A, Smith G D and Rayward-Smith V J 1994 A unified paradigm for parallel genetic algorithms. In: [FOGA94a] pp 131–49
[KAZA95] Kazarlis S A 1995 A genetic algorithm solution to the unit commitment problem, paper presented at: IEEE PES (Power Engineering Society) Winter Meeting, New York, January 1995
[KEAR95] Kearns B 1995 1. A real-life variation of the travelling salesperson problem for several system installers under multiple constraints; 2. Local and wide area network design to maximize performance on three different goals *Genetic Programming List (E-mail List)* 8.3.1995
[KEIJ95] Keijzer M 1995 Optimizing the layout for telecommunication networks in developing countries *Genetic Programming List (E-mail List)* 13.4.1995
[KELL94] Kelly J P, Laguna, M and Glover F 1994 A study of diversification strategies for the quadratic assignment problem *Comput. Operations Res.* **21** 885–93
[KEME95a] van Kemenade C H M, Hendriks C F W, Hesselink H H and Kok J N 1995 Evolutionary computation in air traffic control planning. In: [ESHE95] pp 611–6
[KEME95b] van Kemenade C H M and Kok J N 1995 An evolutionary approach to time constrained routing problems *Report* CS-R 9547, Centrum voor Wiskunde en Informatica, Amsterdam
[KEME95c] van Kemenade C H M 1995 Evolutionary computation in air traffic control planning *Report* CS-R 9550, Centrum voor Wiskunde en Informatica, Amsterdam
[KERS94] Kershaw P S and Edmonds A N 1994 Genetic programming of fuzzy logic production rules with application to financial trading rule generation, paper presented at: ECAI94 Workshop on Applied GAs
[KHUR90] Khuri S and Batarekh A 1990 Genetic algorithms and discrete optimization *'Operations Research 1990', Abstracts of Int. Conf. on Operations Research (Vienna, 28–31 August)* (Vienna: DGOR *et al*) p A147
[KHUR94a] Khuri S, Bäck T and Heitkötter J 1994 The zero/one multiple knapsack problem and genetic algorithms *Proc. 1994 ACM Symp. on Applied Computing* ed E Deaton, D Oppenheim, J Urban and H Berghel (New York: ACM) pp 188–93
[KHUR94b] Khuri S and Bäck T 1994 An evolutionary heuristic for the minimum vertex cover problem. In: [KUNZ94] pp 83–4
[KHUR94c] Khuri S and Bäck T 1994 An evolutionary heuristic for the minimum vertex cover problem. In: [HOPF94] pp 86–90
[KHUR94d] Khuri S, Bäck T and Heitkötter J 1994 An evolutionary approach to combinatorial optimization problems *Proc. 22nd Ann. ACM Computer Science Conf.* ed D Cizmar (New York: ACM) pp 66–73
[KHUR95] Khuri S, Schütz M and Heitkötter J 1995 Evolutionary heuristics for the bin packing problem. In: [PEAR95] pp 285–8
[KIDO93] Kido T, Kitano H and Nakanishi M 1993 A hybrid search for genetic algorithms: combining genetic algorithms, TABU search, and simulated annealing. In: [FORR93] p 641
[KIDW93] Kidwell M D 1993 Using genetic algorithms to schedule distributed tasks on a bus-based system. In: [FORR93] pp 368–74
[KIM94a] Kim G H and Lee C S G 1994 An evolutionary approach to the job-shop scheduling problem *Proc. 1994 IEEE Int. Conf. on Robotics and Automation (San Diego, May 8–13 1994)* vol 1 (Piscataway, NJ: IEEE) pp 501–6
[KIM94b] Kim H, Nara K and Gen M 1994 A method for maintenance scheduling using GA combined with SA *Comput. Ind. Eng.* **27** 477–80

[KIM95] Kim G H and Lee C S G 1995 Genetic reinforcement learning approach to the machine scheduling problem *Proc. IEEE Int. Conf. on Robotics and Automation 1995* (Piscataway, NJ: IEEE) pp 196–201

[KING95] Kingdon J 1995 Intelligent hybrid systems for fraud detection *Proc. 3rd Eur. Conf. on Intelligent Techniques and Soft Computing EUFIT'95 (Aachen)* (Aachen: ELITE Foundation) pp 1135–42

[KINN94] Kinnebrock W 1994 *Optimierung mit Genetischen und Selektiven Algorithmen* (Munich: Oldenbourg)

[KLIM92] Klimasauskas C C 1992 Hybrid neuro-genetic approach to trading algorithms *Advanced Technol. Developers* **1** 1–8

[KNOL93] Knolmayer G and Spahni D 1993 Darstellung und Vergleich ausgewählter Methoden zur Bestimmung von IS-Architekturen *Informatik-Wirtschaft-Gesellschaft, 23 GI-Jahrestagung (Dresden, 27 September–1 Oktober)* ed H Reicher (Berlin: Springer)

[KOBA95] Kobayashi S, Ono I andYamamura M 1995 An efficient genetic algorithm for job shop scheduling problems. In: [ESHE95] pp 506–11

[KOK96] Kok J N 1996 1. Data mining with genetic algorithms; 2. Air traffic flow management with genetic algorithms; talks presented at: 'Evolutionary Algorithms and Their Applications' Seminar, Dagstuhl, Germany, March 1996

[KOPF92] Kopfer H 1992 Konzepte genetischer Algorithmen und ihre Anwendung auf das Frachtoptimierungsproblem im gewerblichen Güterfernverkehr *OR Spektrum* **14** 137–47

[KOPF93a] Kopfer H and Rixen I 1993 Genetische Algorithmen für das Job-Shop Scheduling Problem *Technical Report* 4, Chair of Logistics, Department of Economics, University of Bremen

[KOPF93b] Kopfer H, Erkens E and Pankratz G 1993 Ein Genetischer Algorithmus für das Tourenplanungsproblem, paper presented at: 'Jahrestagung der DGOR und NSOR', Amsterdam 1993

[KOPF94] Kopfer H, Pankratz G and Erkens E 1994 Entwicklung eines hybriden Genetischen Algorithmus zur Tourenplanung *OR Spektrum* **16** 21–31

[KOPF95] Kopfer H, Rixen I and Bierwirth C 1995 Ansätze zur Integration Genetischer Algorithmen in der Produktionsplanung und -steuerung *Wirtschaftsinformatik* **37** 571–80

[KOZA92a] Koza J R 1992 The genetic programming paradigm: genetically breeding populations of computer programs to solve problems. In: [SOUC92] pp 203–321

[KOZA92b] Koza J R 1994 *Genetic Programming. On the Programming of Computers by Means of Natural Selection* (Cambridge, MA: MIT Press)

[KOZA94a] Koza J R 1994 *Genetic Programming II. Automatic Discovery of Reusable Programs* (Cambridge, MA: MIT Press)

[KOZA94b] Koza J R (ed) 1994 *Genetic Algorithms at Stanford 1994* (Stanford: Stanford Bookstore, Stanford University)

[KOZA95] Koza J R 1995 Genetic programming for economic modelling. In: [GOON95a] ch 14

[KOZA96a] Koza J R, Goldberg D E, Fogel D B and Riolo R L (ed) 1996 *Genetic Programming 1996, Proc. 1st Ann. Conf. (Stanford University, July 28–31 1996)* (Cambridge, MA: MIT Press)

[KOZA96b] Koza J R (ed) *Late Breaking Papers at the Genetic Programming 1996 Conf. (Stanford University, July 28–31 1996)* (Stanford: Stanford Bookstore, Stanford University)

[KRAU95] Nissen V and Krause M 1995 On using penalty functions and multicriteria optimisation in facility layout. In: [BIET95a] pp 153–66

[KRES96] Kressel U 1996 Evolutionary algorithms at Daimler-Benz, talk presented at: 'Evolutionary Algorithms and Their Applications' Seminar, Dagstuhl, Germany, March 1996

[KRÖG91] Kröger B, Schwenderling P and Vornberger O 1991 Parallel genetic packing of rectangles. In: [SCHW91] pp 160–4

[KRÖG92] Kröger B, Schwenderling P and Vornberger O 1992 Parallel genetic packing on transputers *Technical Report, Osnabrücker Schriften zur Mathematik* vol I 29, Department of Mathematics/Computer Science, University of Osnabrück

[KUHL95] Kuhl J and Nissen V (ed) 1995 *Evolutionäre Algorithmen in Management-Anwendungen, Proc. Workshop on EA in Management Applications (Göttingen, 23 February 1995)*

[KUNZ94] Kunze J and Stoyan H (ed) 1994 *KI-94 Workshops, 18. Deutsche Jahrestagung für Künstliche Intelligenz (KI-94) (Saarbrücken, 18–23 September 1994)* (Extended Abstracts) (Bonn: Gesellschaft für Informatik)

[KULK92] Kulkarni J and Parsaei H R 1992 Information resource matrix for production and intelligent manufacturing using genetic algorithm techniques *Comput. Ind. Eng.* **23** 483–5

[KURB95a] Kurbel K, Schneider B and Singh K 1995 Parallelization of hybrid simulated annealing and genetic algorithm for short-term production scheduling *Proc. Int. Symp. on Intelligence, Knowledge and Integration for Manufacturing (Southeast University, Nanjing, China, 28–31 March)* ed B Zhong (Nanjing: Southeast University Press) pp 321–6

[KURB95b] Kurbel K and Rohmann T 1995 Ein Vergleich von Verfahren zur Maschinenbelegungsplanung: Simulated Annealing, Genetische Algorithmen und mathematische Optimierung *Wirtschaftsinformatik* **37** 581–93

[KURE96] Kureichick V M, Melikhov A N, Miagkikh V V, Savelev O V and Topchy A P 1996 Some new features in genetic solution of the traveling salesman problem *Proc. ACEDC'96 PEDC (University of Plymouth, UK, 1996)*

[LANE92] Lane D A 1992 Artificial worlds and economics *Working Paper* 92-09-048, Santa Fe Institute, CA

[LANG95] Langdon W B 1995 Scheduling planned maintenance of the national grid. In: [FOGA95c] pp 132–53
[LANG96] Langdon W B 1996 Scheduling maintenance of electric power transmission networks using genetic programming. In: [KOZA96b]
[LASZ90] von Laszewski G 1990 A parallel genetic algorithm for the graph partitioning problem *Transputer Research and Applications 4, Proc. 4th Conf. of the North-American Transputers Users Group* ed D L Fielding (Ithaca: IOS) pp 164–72
[LASZ91] von Laszewski G and Mühlenbein H 1991 Partitioning a graph with a parallel genetic algorithm. In: [SCHW91] pp 165–9
[LAWT92] Lawton G 1992 Genetic algorithms for schedule optimization *AI Expert* **7** 23–7
[LEBA94] LeBaron B 1994 An artificial stock market, talk presented at: MIT AI Vision Seminar Colloquium, Stanford
[LEE93] Lee I, Sikora R and Shaw M J 1993 Joint lot sizing and sequencing with genetic algorithms for scheduling: evolving the chromosome structure. In: [FORR93] pp 383–9
[LEE94] Lee G Y 1994 Genetic algorithms for single-machine job scheduling with common due-date and symmetrical penalties *J. Operations Res. Soc. Japan* **37** 83–95
[LEE95a] Lee G Y 1995 Forecasting time series data in economic systems *Genetic Programming List (E-mail List)* 14.3.1995
[LEE95b] Lee G Y and Choi J Y 1995 A genetic algorithm for job sequencing problems with distinct due and general early–tardy penalty weights *Comput. Operations Res.* **22** 857–69
[LEE95c] Lee G Y and Kim S J 1995 Parallel genetic algorithms for the earliness–tardiness job scheduling problem with general penalty weights *Comput. Ind. Eng.* **28** 231–43
[LENT94] Lent B 1994 Evolution of trade strategies using genetic algorithms and genetic programming. In: [KOZA94b] pp 87–98
[LEON95] Leon V J and Balakrishnan R 1995 Strength and adaptability of problem-space based neighborhoods for resource-constrained scheduling *OR Spektrum* **17** 173–82
[LEU94] Leu Y-Y, Matheson L A and Rees L P 1994 Assembly line balancing using genetic algorithms with heuristic-generated initial populations and multiple evaluation criteria *Decision Sciences* **25** 581–606
[LEVI93a] Levine D M 1993 A genetic algorithm for the set partitioning problem. In: [FORR93] pp 481–7
[LEVI93b] Levine D M 1993 A parallel genetic algorithm for the set partitioning problem *Doctoral Dissertation* Illinois Institute of Technology, IL
[LEVI93c] Levitin G and Rubinovitz J 1993 Genetic algorithm for linear and cyclic assignment problem *Comput. Operations Res.* **20** 575–86
[LEVI95] Levine D 1995 A parallel genetic algorithm for the set partitioning problem *Doctoral Dissertation* Argonne National Laboratory, Argonne, IL
[LEVI96] Levine D 1996 A parallel genetic algorithm for the set partitioning problem *Meta-Heuristics: Theory and Applications* ed I H Osman and J P Kelly (Amsterdam: Kluwer) to appear
[LI90] Li T and Mashford J 1990 A parallel genetic algorithm for quadratic assignment *Proc. ISSM Int. Conf. on Parallel and Distributed Computing and Systems (New York, October 10–12 1990)* ed R A Ammar (New York: Acta) pp 391–4
[LI92] Li Y 1992 Heuristics and exact algorithms for the quadratic assignment problem *Doctoral Dissertation* Pennsylvania State University, PA
[LIAN95] Liang S J T and Mannion M 1995 Scheduling of a flexible assembly system using genetic algorithms *IMechE Conf. Trans.* **3** 173–8
[LIEN94a] Lienig, J and Brandt H 1994 An evolutionary algorithm for the routing of multi-chip modules. In: [DAVI94a] pp 588–97
[LIEN94b] Lienig J and Thulasiraman K 1994 A genetic algorithm for channel routing in VLSI circuits *Evolutionary Computat.* **1** 293–311
[LIEP87] Liepins G E, Hilliard M R, Palmer M and Morrow M 1987 Greedy genetics. In: [GREF87a] pp 90–9
[LIEP90] Liepins G E and Hilliard M R 1990 Genetic algorithms applications to set covering and traveling salesman problems *Operations Research and Artificial Intelligence: The Integration of Problem-Solving Strategies* ed D E Brown and C C White (Dordrecht: Kluwer) pp 29–57
[LIEP91] Liepins G E and Potter W D 1991 A genetic algorithm approach to multiple-fault diagnosis. In: [DAVI91] pp 237–50
[LIM95] Lim S 1995 Job shop scheduling *Genetic Algorithms in Production Scheduling List (E-mail List)* 18.9.1995
[LIND91] Lindgren K 1991 Evolutionary phenomena in simple dynamics *Artificial Life II (Santa Fe Institute Studies in the Sciences of Complexity X)* ed C Langton, C Taylor, J D Farmer and S Rasmussen (Reading, MA: Addison-Wesley)
[LING92a] Ling S E 1992 Integrating a Prolog assignment program with genetic algorithms as a hybrid solution for a polytechnic timetable problem *MSc Dissertation* School of Cognitive and Computing Sciences, University of Sussex, UK
[LING92b] Ling S-E 1992 Integrating genetic algorithms with a PROLOG assignment program as a hybrid solution for a polytechnic timetable problem. In: [MÄNN92] pp 321–9

[LOHB93] Lohbeck T K 1993 Genetic algorithms in layout selection for tree-like pipe networks *MPhil Thesis* University of Exeter, Exeter, UK

[LONT92] Lontke M 1992 Genetischer Algorithmus zur Ein-Depot-Tourenplanung, paper presented at: 'PECOR-Tagung der DGOR', Düsseldorf, 4 December

[LORA95] Loraschi A, Tettamanzi A, Tomassini M and Verda P 1995 Distributed genetic algorithms with an application to portfolio selection problems. In: [PEAR95] pp 384–7

[MACD95] MacDonald C C J 1995 University timetabling problems *Timetabling Problems List (E-mail List)* 11.4.1995

[MACK94] Mack D 1994 Genetische Algorithmen zur Lösung des Klassifikationsproblems. Implementierung eines neuen Genetischen Algorithmus zur Klassifikation (GAK) unter Verwendung des Prototyp-Gedankens *Diploma Thesis* University of Karlsruhe, Germany

[MACK95a] Mack D 1995 Ein Genetischer Algorithmus zur Lösung des Klassifikationsproblems. In: [KUHL95] pp 50–62

[MACK95b] Mackle G, Savic D A and Walters G A 1995 Application of genetic algorithms to pump scheduling for water supply *Genetic Algorithms in Engineering Systems: Innovations and Applications (GALESIA'95) (IEE Conf. Publication No 414)* (Sheffield, UK: IEE) pp 400–5

[MAND91] Manderick B, Weger M D and Spiessens P 1991 The genetic algorithm and the structure of the fitness landscape. In: [BELE91] pp 143–50

[MANI91] Maniezzo V 1991 The rudes and the shrewds: an experimental comparison of several evolutionary heuristics applied to the QAP problem *Report* 91-042, Department of Electronics, Milan Polytechnic, Italy

[MANI95] Maniezzo V, Colorni A and Dorigo M 1995 ALGODESK: An experimental comparison of eight evolutionary heuristics applied to the QAP problem *Eur. J. Operational Res.* **81** 188–204

[MÄNN92] Männer R and Manderick B (ed) 1992 *Parallel Problem Solving from Nature, Proc. 2nd Conf. on Parallel Problem Solving from Nature (Brussels, 28–30 September)* (Amsterdam: North-Holland)

[MANS91] Mansour N and Fox G C 1991 A hybrid genetic algorithm for task allocation in multicomputers. In: [BELE91] pp 466–73

[MAO95] Mao J Z and Wu Z M 1995 Genetic algorithm and the application for job-shop group scheduling *Proc. SPIE* vol 2620 (Bellingham, WA: Society of Photo-Optical Instrumentation Engineers) pp 103–8

[MARE92a] Marengo L 1992 *Structure, Competence and Learning in an Adaptive Model of the Firm (Papers on Economics and Evolution 9203)* ed European Study Group for Evolutionary Economics (Freiburg: European Study Group for Evolutionary Economics)

[MARE92b] Marengo L 1992 Coordination and organizational learning in the firm *J. Evolutionary Econ.* **2** 313–26

[MARE94a] Maresky J, Davidor Y, Gitler D, Aharoni G and Barak A 1994 Selectively destructive restart. In: [KUNZ94] pp 71–2

[MARE94b] Maresky J, Davidor Y, Gitler D, Aharoni G and Barak A 1994 Selectively destructive restart. In: [HOPF94] pp 50–5

[MARE95] Maresky J, Davidor Y, Gitler D, Aharoni G and Barak A 1995 Selectively destructive re-start. In: [ESHE95] pp 144–50

[MARG91] Margarita S 1991 Neural networks, genetic algorithms and stock trading *Artificial Neural Networks, Proc. 1991 Int. Conf. in Artificial Neural Networks (ICANN-91) (Espoo, Finland, 24–28 June)* ed T Kohonen (Amsterdam: North-Holland) pp 1763–6

[MARG92] Margarita S 1992 Genetic neural networks for financial markets: some results *Proc. 10th Eur. Conf. on Artificial Intelligence (Vienna, 3–7 August)* ed B Neumann (Chichester: Wiley) pp 211–3

[MARI94] Marin F J, Trelles-Salazar O and Sandoval F 1994 Genetic algorithms on LAN-message passing architectures using PVM: application to the routing problem. In: [DAVI94a] pp 534–43

[MARK89] Marks R E 1989 Breeding hybrid strategies: optimal behavior for oligopolists. In: [SCHA89] pp 198–207

[MARK92a] Marks R E 1992 Repeated games and finite automata *Recent Advances in Game Theory* ed J Creedy, J Eichberger and J Borland (London: Arnold) pp 43–64

[MARK92b] Marks R E 1992 Breeding optimal strategies: optimal behaviour for oligopolists *J. Evolutionary Econ.* **2** 17–38

[MARK95] Marks R E, Midgley D F and Cooper L G 1995 Adaptive behaviour in an oligopoly. In: [BIET95a] pp 225–39

[MARU92] Maruyama T, Konagaya A and Konishi K 1992 An asynchronous fine-grained parallel genetic algorithm. In: [MÄNN92] pp 563–72

[MARU93] Maruyama T, Hirose T and Konagaya A 1993 A fine-grained parallel genetic algorithm for distributed parallel systems. In: [FORR93] pp 184–90

[MATH92] Mathias K and Whitley D 1992 Genetic operators, the fitness landscape and the traveling salesman problem. In: [MÄNN92] pp 219–28

[MATS90] Matsuo K and Adachi N 1990 Metastable antagonistic equilibrium and stable cooperative equilibrium in distributed prisoner's dilemma game *Advances in Support Systems Research* ed G E Lasker and R R Hough (Windsor, Ontario: International Institute for Advanced Studies in Systems Research and Cybernetics) pp 483–7

[MATT94] Mattfeld D C, Kopfer H and Bierwirth C 1994 Control of parallel population dynamics by social-like behavior of GA-individuals. In: [DAVI94a] pp 16–25

[MATT96] Mattfeld D C 1996 Evolutionary search and the job shop: investigations on genetic algorithms for production scheduling (Heidelberg: Physica)

[MATW91] Matwin S, Szapiro S and Haigh K 1991 Genetic algorithms approach to negotiation support system *IEEE Trans. Syst., Man Cybern.* **21** 102–14

[MAYR95] Mayrand É, Lefrançois P, Kettani O and Jobin M-H 1995 A genetic search algorithm to optimize job sequencing under a technological constraint in a rolling-mill facility *OR Spektrum* **17** 183–91

[MAZA93] de la Maza M and Tidor B 1993 An analysis of selection procedures with particular attention paid to proportional and Boltzmann selection. In: [FORR93] pp 124–31

[MAZA95] de la Maza M and Yuret D 1995 A model of stock market participants. In: [BIET95a] pp 290–304

[MAZI92] Maziejewski S 1992 The vehicle routing and scheduling problem with time window constraints using genetic algorithms *Diploma Thesis* Department of Logic, Complexity and Deduction, University of Karlsruhe (TH), Germany. Also Norges Tekniske Høgskole, Institutt for Datateknikk og Telematik, Norway

[MAZI93] Maziejewski S 1993 Ein Genetischer Ansatz für die Tourenplanung mit Kundenzeitschranken, paper presented at: 'Jahrestagung der DGOR und NSOR', Amsterdam

[MCDO95a] McDonnell J R, Reynolds R G and Fogel D B (ed) 1995 *Proc. 4th Ann. Conf. on Evolutionary Programming (EP 95) (San Diego, CA, March 1–3 1995* (Cambridge, MA: MIT Press)

[MCDO95b] McDonnell J R, Fogel D B, Rindt C R, Recker W W and Fogel L J 1995 Using evolutionary programming to control metering rates on freeway ramps. In: [BIET95a] pp 305–27

[MCMA95] McMahon G and Hadinoto D 1995 Comparison of heuristic search algorithms for single machine scheduling problems. In: [YAO95a] pp 293–304

[MENO95] Menon A, Mehrotra K, Mohan C K and Ranka S 1995 Optimization using replicators. In: [ESHE95] pp 209–16

[MESM95] Mesman B 1995 Job shop scheduling *Genetic Algorithms in Production Scheduling List (E-mail List)* 18.9.1995

[MICH94] Michalewicz Z and Arabas J 1994 Genetic algorithms for the 0/1 knapsack problem *Methodologies for Intelligent Systems (Lecture Notes in Artificial Intelligence 869)* ed Z W Ras and M Zemankowa (Berlin: Springer) pp 134–43

[MICH95] Micheletti A 1995 Short term production scheduling *Genetic Algorithms in Production Scheduling List (E-mail List)* 27.4.1995

[MICH96] Michalewicz Z 1996 *Genetic Algorithms + Data Structures = Evolution Programs* (Berlin: Springer) 3rd edn

[MILL89] Miller J 1989 The coevolution of automata in the repeated prisoner's dilemma *Working Paper* 89-003, Santa Fe Institute, CA

[MILL93] Mills G 1993 Scheduling trains, personal communication

[MOCC95] Mocci F 1995 Crew-scheduling in an industrial plant *Timetabling Problems List (E-Mail List)* 29.3.1995

[MOOR94] Moore B 1994 Effect of population replacement strategy on accuracy of genetic algorithm market forecaster. In: [HILL94] pp 169–78

[MORI92] Morikawa K, Furuhashi T and Uchikawa Y 1992 Single populated genetic algorithm and its application to jobshop scheduling *Proc. 1992 Int. Conf. on Industrial Electronics, Control, and Instrumentation (San Diego, CA, November 9–13 1992* vol 2 (Piscataway, NJ: IEEE) pp 1014–8

[MOSC92] Moscato P and Norman M G 1992 A 'memetic' approach for the traveling salesman problem. Implementation of a computational ecology for combinatorial optimization on message-passing systems *Proc. PACTA'92—Int. Conf. on Parallel Computing and Transputer Applications (Barcelona, 21–25 September 1992)* ed M Valero (Barcelona: CIMNE *et al*)

[MÜHL87] Mühlenbein H, Gorges-Schleuter M and Krämer O 1987 New solutions to the mapping problem of parallel systems: the evolution approach *Parallel Comput.* **4** 269–79

[MÜHL88] Mühlenbein H, Gorges-Schleuter M and Krämer O 1988 Evolution algorithms in combinatorial optimization *Parallel Comput.* **7** 65–85

[MÜHL89] Mühlenbein H 1989 Parallel genetic algorithms, population genetics and combinatorial optimization. In: [SCHA89] pp 416–21

[MÜHL90] Mühlenbein H 1990 Parallel genetic algorithms, population genetics and combinatorial optimization *Evolution and Optimization'89. Selected Papers on Evolution Theory, Combinatorial Optimization, and Related Topics* ed H-M Voigt, H Mühlenbein and H-P Schwefel (Berlin: Akademie) pp 79–85

[MÜHL91a] Mühlenbein H 1991 Parallel genetic algorithms, population genetics and combinatorial optimization. In: [BECK91] pp 398–406

[MÜHL91b] Mühlenbein H 1991 Evolution in time and space—the parallel genetic algorithm. In: [RAWL91] pp 316–37

[MÜHL91c] Mühlenbein H 1991 Darwin's continent cycle theory and its simulation by the prisoner's dilemma *Complex Syst.* **5** 459–78

[MÜHL92] Mühlenbein H 1992 Parallel genetic algorithms and combinatorial optimization *Proc. Computer Science and Operations Research—New Developments in their Interfaces* (Oxford: Pergamon)

[MULK93] Mulkens H 1993 Revisiting the Johnson algorithm for flowshop scheduling with genetic algorithms *Knowledge-Based Reactive Scheduling (Athens, 1 October 1993)* ed E Ezelke and R M Kerr *IFIP Trans. B: Applications Technol.* **15** 69–80

[MÜLL83a] Müller H and Pollhammer G 1983 Evolutionsstrategische Lastflussoptimierung *Technical Report* p5068, Department for Electrics and High Voltage Technology, Technical University of Berlin

[MÜLL83b] Müller H 1983 Power flow optimization in electric networks by evolutionary strategic search *Proc. 6th Eur. Congress on Operations Research (July 1983)* (Amsterdam: Association of European Operational Research Societies) p 107

[MÜLL86] Müller H, Theil G and Waldmann W 1986 Results of evolutional random search procedure for load flow optimization in electric networks *System Modelling and Optimization, Proc. 12th IFIP Conf. 1985* ed A Prekopa, J Szelezsan and B Strazicky (Berlin: Springer) pp 628–36

[MUNA93] Munakata T and Hashier D J 1993 A genetic algorithm applied to the maximum flow problem. In: [FORR93] pp 488–93

[MURA94] Murata T and Ishibuchi H 1994 Performance evaluation of genetic algorithms for flowshop scheduling problems *Proc. 1st IEEE Conf. on Evolutionary Computing (Orlando, FL, June 27–29 1994)* vol 2 (Piscataway, NJ: IEEE) pp 812–17

[MURP92] Murphy L J and Simpson A R 1992 Genetic algorithms in pipe network optimization *Research Report* R93, Department of Civil and Environmental Engineering, University of Adelaide, Australia

[MURP93] Murphy L J, Simpson A R and Dandy G C 1993 Pipe network optimization using an improved genetic algorithm *Research Report* R109, Department of Civil and Environmental Engineering, University of Adelaide

[MURR96] Murray J 1996 Item scheduling in a steel flamecutting operation *Genetic Algorithms in Production Scheduling List (E-mail List)* 2.7.1996

[NACH93] Nachtigall K and Voget S 1993 A genetic algorithm approach to periodic programs *Technical Report, Hildesheimer Informatik-Berichte* 16/93, Department of Mathematics, University of Hildesheim

[NACH95a] Nachtigall K and Voget S 1995 Optimierung von periodischen Netzplänen mit Genetischen Algorithmen. In: [KUHL95] pp 41–9

[NACH95b] Nachtigall K and Voget S 1995 Optimierung von Periodischen Netzplänen mit Genetischen Algorithmen *Technical Report, Hildesheimer Informatik-Berichte* 1/95, Department of Mathematics, University of Hildesheim

[NACH95c] Nachtigall K and Voget S 1995 Optimization of the intergrated fixed interval timetable *Technical Report, Hildesheimer Informatik-Berichte* 20/95, Department of Mathematics, University of Hildesheim

[NACH96] Nachtigall K and Voget S 1996 A genetic algorithm approach to periodic railway synchronization *Comput. Operations Res.* **23** 453–63

[NAKA91] Nakano R and Yamada T 1991 Conventional genetic algorithm for job shop problems. In: [BELE91] pp 474–79

[NAKA94] Nakano R, Davidor Y and Yamada T 1994 Optimal population size under constant computation cost. In: [DAVI94a] pp 130–8

[NAPI89] Napierala G 1989 Ein paralleler Ansatz zur Lösung des TSP *Diploma Thesis* University of Bonn, Germany

[NETT93] Nettleton D J and Garigliano R 1993 Large ratios of mutation to crossover: the example of the travelling salesman problem *Adaptive and Learning Systems II (Orlando, FL, April 12–13 1993)* vol SPIE-1962, ed F A Sadjadi (Bellingham, WA: Society of Photo-Optical Instrumentation Engineers) pp 110–9

[NEUH91] Neuhaus P 1991 Solving the mapping-problem—experiences with a genetic algorithm. In: [SCHW91] pp 170–5

[NEWE69] Newell A 1969 Heuristic programming: ill structured problems *Progress in Operations Research—Relationships between Operations Research and the Computer* vol 3, ed J Aronofsky (New York: Wiley) pp 361–414

[NG95] Ng K P and Wong K C 1995 A new diploid scheme and dominance change mechanism for non-stationary function optimization. In: [ESHE95] pp 159–66

[NISH91] Nishikawa Y and Tamaki H 1991 A genetic algorithm as applied to the jobshop scheduling *Trans. Soc. Instrum. Control Engrs (Japan)* **27** 593–9 (in Japanese)

[NISS92] Nissen V 1992 Evolutionary algorithms for the quadratic assignment problem *Technical Report* Computer Science Department, University of Göttingen

[NISS93a] Nissen V 1993 A new efficient evolutionary algorithm for the quadratic assignment problem *Operations Research Proc., Papers of the 21st Ann. Meeting of DGOR in Cooperation with ÖGOR 1992 (Aachen, 9–11 September 1992)* (Berlin: Springer) pp 259–67

[NISS93b] Nissen V 1993 *Evolutionary Algorithms in Management Science. An Overview and List of References (Papers on Economics and Evolution 9303)* ed European Study Group for Evolutionary Economics (Freiburg: European Study Group for Evolutionary Economics)

[NISS94a] Nissen V 1994 *Evolutionäre Algorithmen. Darstellung, Beispiele, betriebswirtschaftliche Anwendungsmöglichkeiten* (Wiesbaden: DUV)

[NISS94b] Nissen V and Krause M 1994 Constrained combinatorial optimization with an evolution strategy *Fuzzy Logik. Theorie und Praxis* ed B Reusch (Berlin: Springer) pp 33–40
[NISS94c] Nissen V 1994 A comparison of different evolutionary algorithms on the quadratic assignment problem. In: [KUNZ94] pp 75–6
[NISS94d] Nissen V 1994 Comparing different evolutionary algorithms on the quadratic assignment problem. In: [HOPF94] pp 55–72
[NISS94e] Nissen V 1994 Solving the quadratic assignment problem with clues from Nature *IEEE Trans. Neural Networks* Special Issue on evolutionary computation **5** 66–72
[NISS95a] Nissen V 1995 Evolutionary optimisation of a stochastic inventory simulation *Operations Research Proc. 1994. Selected Papers of the Int. Conf. on Operations Research (Berlin, 30 August–2 September 1994)* ed U Derigs, A Bachem and A Drexl (Berlin: Springer) pp 466–71
[NISS95b] Nissen V and Biethahn J 1995 Determining a good inventory policy with a genetic algorithm. In: [BIET95a] pp 240–50
[NISS95c] Nissen V and Biethahn J 1995 Zur Robustheit Genetischer Algorithmen bei der Stochastischen Optimierung am Beispiel der Simulation. In: [BIET95b] pp 113–31
[NISS95d] Nissen V and Paul H 1995 A modification of threshold accepting and its application to the quadratic assignment problem *OR Spektrum* **17** 205–10
[NISS95e] Nissen V 1995 An overview of evolutionary algorithms in management applications. In: [BIET95a] pp 44–97
[NOBL90] Noble A 1990 Using genetic algorithms in financial services *Proc. Two Day Int. Conf. on Forecasting and Optimisation in Financial Services (3–4 October)* (London: Technical Services, IBC Ltd)
[NOCH86] Noche B, Kottkamp R, Lücke G and Peters E 1986 Optimizing simulators within material flow systems *Proc. 2nd Eur. Simulation Congress Simulation Councils* (Ghent, Belgium: SCS Europe) pp 651–7
[NOCH90] Noche B 1990 Simulation in Produktion und Materialfluss *Entscheidungsorientierte Simulationsumgebung* (Cologne: TÜV Rheinland)
[NORD94] Nordstrom A L and Tufekci S 1994 A genetic algorithm for the talent scheduling problem *Comput. Operations Res.* **21** 927–40
[NORM95] Norman B A 1995 Job shop scheduling *Genetic Algorithms Digest (E-mail List)* 9 (26) 1.5.1995
[NOTT92] Nottola C, Leroy F and Davalo F 1992 Dynamics of artificial markets *Toward a Practice of Autonomous Systems, Proc. 1st Eur. Conf. on Artificial Life* ed F J Varela and P Bourgine (Cambridge, MA: MIT Press) pp 185–94
[NYGA90] Nygard K E and Kadaba N 1990 Modular neural networks and distributed adaptive search for traveling salesman algorithms *Applications of Artificial Neural Networks (Proc. SPIE vol 1294)* (Bellingham, WA: Society of Photo-Optical Instrumentation Engineers) pp 442–51
[NYGA92] Nygard K E and Yang C-H 1992 Genetic algorithms for the travelling salesman problem with time windows *Computer Science and Operations Research. New Developments in their Interfaces (Williamsburg, January 8–10 1992)* ed O Balci, R Sharda and S A Zenios (Oxford: Pergamon) pp 411–23
[OLIV87] Oliver I M, Smith D J and Holland J R C. 1987 A study of permutation crossover operators on the traveling salesman problem. In: [GREF87a] pp 224–30
[OLIV93] Oliver J R 1993 Discovering individual decision rules: an application of genetic algorithms. In: [FORR93] pp 216–22
[OMAN96] Oman S 1996 Job shop scheduling *Genetic Algorithms in Production Scheduling List (E-mail List)* 21.3.1996
[ORDI95] Ordieres Mere J B 1995 Best cut bar problem to minimize waste material *Genetic Algorithms in Production Scheduling List (E-mail List)* 30.1.195
[ORER96] Orero S O and Irving M R 1996 A genetic algorithm for generator scheduling in power-systems *Int. J. Electr. Power Energy Syst.* **18** 19–26
[OSTE92] Ostermeier A 1992 An evolution strategy with momentum adaption of the random number distribution. In: [MÄNN92] pp 197–206
[OSTE94] Ostermeier A, Gawelczyk A and Hansen N 1994 Step-size adaptation based on non-local use of selection information. In: [DAVI94a] pp 189–98
[OUSS96] Oussaidène M, Chopard B, Pictet O V and Tomassini M 1996 Parallel genetic programming: an application to trading models evolution. In: [KOZA96a]
[PACK90] Packard N H 1990 A genetic learning algorithm for the analysis of complex data *Complex Systems* **4** 543–72
[PAEC93] Paechter B, Lucian H and Cumming A 1993 An evolutionary approach to the general timetable problem *Sci. Ann. Al I Cuza University of Iasi (Romania)* Special Issue for the ROSYCS Symposium
[PAEC94a] Paechter B, Cumming A, Luchian H and Petriuc M 1994 Two solutions to the general timetable problem using evolutionary methods *Proc. IEEE Conf. on Evolutionary Computation 1994* (Piscataway, NJ: IEEE)
[PAEC94b] Paechter B 1994 Optimising a presentation timetable using evolutionary algorithms. In: [FOGA94a] pp 264–76

[PAEC95] Paechter B, Cumming A and Luchian H 1995 The use of local search suggestion lists for improving the solution of timetable problems with evolutionary algorithms. In: [FOGA95c] pp 86–93

[PALM94a] Palmer C C 1994 Application of GP techniques to various graph theoretic problems *Genetic Programming List (E-mail List)* 5.8.1994

[PALM94b] Palmer G 1994 An integrated approach to manufacturing planning *PhD Thesis* University of Huddersfield, UK

[PANK93] Pankratz G 1993 Entwicklung, Realisierung und Konfigurierung eines Genetischen Algorithmus für das Standard-Tourenplanungsproblem *Diploma Thesis* University of Siegen, Germany

[PARA95] Parada Daza V, Muñoz R and Gómes de Alvarenga A 1995 A hybrid genetic algorithm for the two-dimensional guillotine cutting problem. In: [BIET95a] pp 183–96

[PARE92a] Paredis J 1992 Exploiting constraints as background knowledge for genetic algorithms: a case-study for scheduling. In: [MÄNN92] pp 229–38

[PARE92b] Paredis J and Van Rij T 1992 Intelligent modelling, simulation and scheduling of discrete production processes *Proc. Computer Science and Operations Research—New Developments in their Interfaces* (Oxford: Pergamon)

[PARE93] Paredis J 1993 Genetic state-space search for constrained optimization problems *IJCAI-93, Proc. 13th Int. Joint Conf. on Artificial Intelligence (Chambéry, 28 August–3 September 1993)* vol 2 (San Mateo, CA: Morgan Kaufmann) pp 967–72

[PARK95a] Park K and Carter B 1995 On the effectiveness of genetic search in combinatorial optimization *Proc. 10th ACM Symp. on Applied Computing, GA and Optimization Track* (New York: ACM) pp 329–36

[PARK95b] Park K 1995 A comparative study of genetic search. In: [ESHE95] pp 512–9

[PARK95c] Park L-J 1995 Job shop scheduling *Genetic Algorithms in Production Scheduling List (E-mail List)* 20.4.1995

[PARK95d] Park L-J 1995 Job shop scheduling *Genetic Algorithms in Production Scheduling List (E-mail List)* 5.7.1995

[PARK95e] Park L-J and Park C H 1995 Genetic algorithm for job-shop scheduling problems based on two representational schemes *Electron. Lett.* **31** 2051–3

[PEAR95] Pearson D W, Steele N C and Albrecht R F (ed) 1995 Artificial neural nets and genetic algorithms *Proc. Int. Conf. (Alès, France)* (Berlin: Springer)

[PERR94] Perry J E 1994 The effect of population enrichment in genetic programming *Proc. IEEE World Congress on Computational Intelligence* (Piscataway, NJ: IEEE)

[PESC92] Pesch E and Dorndorf U 1992 Job Shop Scheduling mittels Genetischer Algorithmen, paper presented at: 'Jahrestagung der DGOR/ÖGOR', Aachen, 9–11 September

[PESC93] Pesch E 1993 Machine learning by schedule decomposition *Research Memorandum* 93-045, Limburg University, Belgium

[PESC94] Pesch E 1994 *Learning in Automated Manufacturing* (Heidelberg: Physica)

[PETE90] Peterson C 1990 Parallel distributed approaches to combinatorial optimization: benchmark studies on travelling salesman problem *Neural Computat.* **2** 261–9

[PICO94] Pico C A G. and Wainwright R L 1994 Dynamic scheduling of computer tasks using genetic algorithms *Proc. IEEE Conf. on Evolutionary Computation* vol 2 (Piscataway, NJ: IEEE) pp 829–33

[PICT95] Pictet O V, Dacorogna M M, Chopard B, Oussaidene M, Schirru R and Tomassini M 1995 Using genetic algorithms for robust optimization in financial applications *Neural Network World* issue 4/95, pp 573–87

[PIGG95] Piggott P and Suraweera F 1995 Encoding graphs for genetic algorithms: an investigation using the minimum spanning tree problem. In: [YAO95a] pp 305–14

[POON90] Poon P W 1990 Genetic algorithms and fuel cycle optimization *Nucl. Engineer* **31** 173–7

[POON92] Poon P W and Parks G T 1992 Optimising PWR reload core designs. In: [MÄNN92] pp 371–80

[PORT95] Portmann M-C 1995 Flowshop scheduling with GA and branch&bound techniques *Genetic Algorithms in Production Scheduling List (E-mail List)* 9.9.1995

[POTV96] Potvin J Y and Guertin F 1996 The clustered traveling salesman problem: a genetic approach *Meta-Heuristics: Theory and Applications* ed I H Osman and J P Kelly (Dordrecht: Kluwer) to appear

[PRIN93] Prinetto P, Rebaudengo M and Reorda M S 1993 Hybrid genetic algorithms for the travelling salesman problem. In: [ALBR93] pp 559–66

[PROS88] Prosser P 1988 A hybrid genetic algorithm for pallet loading *Proc. Eur. Conf. on AI (Munich)* (London: Pitman) pp 159–64

[QUEI95] Queiros F 1995 GA to solve timetabling problem *Timetabling Problems List (E-Mail List)* 26.6.1995

[QUIN95] Quinn G R 1995 Two-stage production scheduling problem, personal communication

[RAWL91] Rawlins G J E (ed) 1991 *Foundations of Genetic Algorithms* (San Mateo, CA: Morgan Kaufmann)

[RECH73] Rechenberg I 1973 *Evolutionsstrategie. Optimierung technischer Systeme nach Prinzipien der biologischen Evolution* (Stuttgart: Frommann-Holzboog)

[REEV91] Reeves C R 1991 Recent algorithmic developments applied to scheduling problems *Proc. 9th IASTED Symp. on Applied Informatics (Innsbruck, Austria, February 18–21 1991* (Anaheim: ACTA) pp 155–8

[REEV92a] Reeves C R 1992 A genetic algorithm approach to stochastic flowshop sequencing *Digest* 1992/106, 8 May 1992 (London: IEE) pp 131–4

[REEV92b] Reeves C R 1992 A genetic algorithm approach to stochastic flowshop sequencing *Proc. IEE Colloq. on Genetic Algorithms for Control and Systems Engineering (London, 1992)* (London: IEE)

[REEV93] Reeves C R 1993 Bin packing, personal communication

[REEV95] Reeves C R 1995 A genetic algorithm for flowshop sequencing *Comput. Operations Res.* **22** 5

[REPP85] Reppenhagen R 1985 Adaptive Suchalgorithmen *Diploma Thesis* Department of Mathematics/Computer Science, University of Paderborn, Germany

[RIEC94] Rieck C 1994 Evolutionary simulation of asset trading strategies. In: [HILL94] pp 112–36

[RIXE95] Rixen I, Bierwirth C and Kopfer H 1995 A case study of operational just-in-time scheduling using genetic algorithms. In: [BIET95a] pp 112–23

[ROBB95] Robbins P 1995 The use of a variable length chromosome for permutation manipulation in genetic algorithms. In: [PEAR95] pp 144–147

[ROBI95] Robinson G and McIlroy P 1995 Exploring some commercial applications of genetic programming. In: [FOGA95c] pp 234–64

[ROGN94] Rognerud N 1994 Building financial trading models, personal communication

[ROGN95] Rognoni R 1995 Job shop scheduling *Genetic Algorithms in Production Scheduling List (E-mail List)* 9.3.1995

[ROSS94a] Ross P, Corne D and Fang H-L 1994 Successful lecture timetabling with evolutionary algorithms, *Workshop Notes, ECAI'94 Workshop W17: Applied Genetic and other Evolutionary Algorithms* ed A E Eiben, B Manderick and Z Ruttkay

[ROSS94b] Ross P, Corne D and Fang H-L 1994 Improving evolutionary timetabling with delta evaluation and directed mutation *Research Paper* 707, Department of Artificial Intelligence, University of Edinburgh, Edinburgh, UK

[ROSS94c] Ross P, Corne D and Fang H-L 1994 Improving evolutionary timetabling with delta evaluation and directed mutation. In: [DAVI94a] pp 556–65

[ROSS95] Ross P and Corne D 1995 Comparing genetic algorithms, simulated annealing, and stochastic hillclimbing on timetabling problems. In: [FOGA95c] pp 94–102

[RUBI92] Rubin P A and Ragatz G L 1992 Scheduling in a sequence dependent setup environment with genetic search *Proc. Ann. Meeting of the Decision Science Institute (San Francisco, November 22–24)* (Philadelphia, PA: American Institute for Decision Sciences) pp 1029–31

[RUBI95] Rubin P A and Ragatz G L 1995 Scheduling in a sequence-dependent setup environment with genetic search *Comput. Operations Res.* **22** 85–99

[RUDO91] Rudolph G 1991 Global optimization by means of distributed evolution strategies. In: [SCHW91] pp 209–13

[RUDO94] Rudolph G 1994 An evolutionary algorithm for integer programming. In: [DAVI94a] pp 139–48

[SAKA93] Sakawa M, Utaka J, Inuiguchi M, Shiromaru I, Suginohara N and Inoue T 1993 Hot parts operating schedule of gas turbines by genetic algorithms and fuzzy satisficing methods *Proc. 1993 Int. Joint Conf. on Neural Networks* vols 1–3 (Piscataway, NJ: IEEE) pp 746–9

[SALE94] Salewski R and Bartsch T 1994 A comparison of genetic and greedy randomized algorithms for medium-to short-term audit-staff scheduling *Working Paper* University of Kiel

[SANG95] Sanagalli M and Semeraro Q and Tolio T 1995 Performance of genetic algorithms in the solution of permutation flowshop problems. In: [PEAR95] pp 495–8

[SANN88] Sannier A V and Goodman E D 1988 Midgard: A genetic approach to adaptive load balancing for distributed systems *Proc. 5th Int. Conf. on Machine Learning (University of Michigan, Ann Arbor, MI, June 12–14)* ed J Laird (San Mateo, CA: Morgan Kaufmann) pp 174–80

[SANN94] Sannomiya N, Iima H and Akatsuka N 1994 Genetic algorithm approach to a production ordering problem in an assembly process with constant use of parts *Int. J. Syst. Sci.* **25** 1461–72

[SAVI94a] Savic D A and Walters G A 1994 Genetic algorithms and evolution program decision support ed Knezevic J *Proc. Int. Symp. on Advances in Logistics (University of Exeter)* (Exeter: University of Exeter Press) pp 72–80

[SAVI94b] Savic D A and Walters G A 1994 Evolution programs in optimal design of hydraulic networks *Adaptive Computing in Engineering Design and Control (University of Plymouth, UK)* ed I C Parmee (Plymouth, UK: University of Plymouth) pp 146–50

[SAVI95a] Savic D A and Walters G A 1995 Place of evolution programs in pipe network optimization *Integrated Water Resources Planning for the 21st Century (ASCE National Conf. on Water Resources Planning and Management, Boston, MA, May 7–10)* ed M F Domenica (New York: ASCE) pp 592–5

[SAVI95b] Savic D A and Walters G A 1995 Genetic operators and constraint handling for pipe network optimization. In: [FOGA95c] pp 154–65

[SAVI95c] Savic D A, Walters G A and Knezevic J 1995 Optimal opportunistic maintenance policy using genetic algorithms, 1: formulation *J. Quality Management Eng.* **1** (2) 34–49

[SAVI95d] Savic D A, Walters G A and Knezevic J 1995 Optimal opportunistic maintenance policy using genetic algorithms, 2: analyses *J. Quality Management Eng.* **1** (3) 25–34

[SAVI95e] Savic D A and Walters G A 1995 An evolution program for optimal pressure regulation in water distribution networks *Eng. Optimization* **24** 197–219

[SAVI96] Savic D A and Walters G A 1996 Integration of a model for hydraulic analysis of water distribution networks with an evolution program for pressure regulation *Microcomput. Civ. Eng.* **11** (2) 87–97

[SCHA89] Schaffer J D (ed) 1989 *Proc. 3rd Int. Conf. on Genetic Algorithms (George Mason University, June 4–7)* (San Mateo, CA: Morgan Kaufmann)

[SCHA94] Schack B 1994 Scheduling in a liquid petroleum pipeline, personal communication

[SCHI81] Schiemangk C 1981 Anwendung einer Evolutionsstrategie zum Auffinden eines optimalen Subgraphen *Numerische Realisierung mathematischer Modelle* ed Zingert (Berlin: Akademie der Wissenschaften der DDR) pp 167–81

[SCHL96] Schlichtmann U 1996 Genetische Algorithmen zur Lösung von Tourenplanungsproblemen *Diploma Thesis* Fachbereich Wirtschaftswissenschaften, Fernuniversität Hagen, Germany

[SCHO76] Schoebel R 1976 Anwendung der Evolutionsstrategie auf deterministische Modelle der Operations Research *Working Paper* Department of Bionics, Technical University of Berlin

[SCHÖ90] Schöneburg E, Heinzmann F and Dörrich T 1990 Produktionsplanung mit Methoden der Künstlichen Intelligenz und Genetischen Algorithmen *CHIP Professional* **9** 68–75

[SCHÖ91] Schöneburg E, Heinzmann F and Dörrich T 1991 Industrielle Planung mit Methoden der künstlichen Intelligenz *DV-Management* **1** 26–9

[SCHÖ92] Schöneburg E and Heinzmann F 1992 PERPLEX: Produktionsplanung nach dem Vorbild der Evolution *Wirtschaftsinformatik* **34** 224–32

[SCHO93] Schoenauer M and Xanthakis S 1993 Constrained GA optimization. In: [FORR93] pp 573–80

[SCHÖ93] Schöneburg E 1993 Zeitreihenanalyse und -prognose mit Evolutionsalgorithmen *Working Paper* Expert Informatik GmbH, Berlin

[SCHÖ94] Schöneburg E, Heinzmann F and Feddersen S 1994 *Genetische Algorithmen und Evolutionsstrategien. Eine Einführung in Theorie und Praxis der simulierten Evolution* (Bonn: Addison-Wesley)

[SCHR91] Schroetter M 1991 Planung des Werkfernverkehrs mit Genetischen Algorithmen *Diploma Thesis* Department of Computer Science, University of Erlangen-Nürnberg

[SCHÜ92] Schütz G 1992 Ein genetischer Algorithmus zum Bandabgleich, paper presented at: 'PECOR-Tagung der DGOR', Düsseldorf, 4 December

[SCHU93a] Schuh H and von Randow G 1993 Gut erforscht ist halb erfunden *Die Zeit* **9** 42 (26 February 1993)

[SCHU93b] Schulte J and Becker B-D 1993 Optimierung in der Werkstattsteuerung: Simulation und Genetische Algorithmen *Proc. 8. Symp. Simulationstechnik (ASIM, Berlin, September 1993)* (Braunschweig: GI-ASIM) pp 599–602

[SCHU93c] Schulte J and Becker B-D 1993 Production scheduling using genetic algorithms *Selected Papers from the 7th IFAC/IFIP/IFORS/IMACS/ISPE Symp. (Toronto, May 25–28 1992)* (Oxford: Pergamon) pp 367–72

[SCHÜ94] Schüler T and Uthmann T 1994 Erzeugung realitätsnaher Produktionspläne durch Einsatz Genetischer Algorithmen (GAs). In: [KUNZ94] pp 230–1

[SCHU94] Schulte J and Frank A 1994 Optimierung von Losgrösse und Reihenfolge einer Serienfertigung: Gegenüberstellung einer Heuristik mit den Genetischen Algorithmen *Jahrbuch Fraunhofer-Institut für Produktionstechnik und Automatisierung* (Stuttgart: Fraunhofer-Institut für Produktionstechnik und Automatisierung)

[SCHÜ95a] Schütz G 1995 Parallele Suche zur Lösung mehrstufiger Standortprobleme (Abstract). In: [KUHL95] p 40

[SCHÜ95b] Schütz G 1995 Ein kooperativer Suchansatz zur Lösung mehrstufiger Distributionsprobleme, paper presented at: Symposium Operations Research SOR'95, Passau, Germany, 13–15 September 1995

[SCHÜ95c] Schütz G 1995 Verteilte parallele Graphensuche am Beispiel mehrstufiger kapazitierter 'warehouse location'-Probleme, paper presented at: Symposium Operations Research SOR'95, Passau, Germany, 13–15 September 1995

[SCHW72] Schwefel D 1972 Gesundheitsplanung im departamento del valle del cauca *Report of the German Development Institute, Berlin* (Berlin: German Development Institute)

[SCHW81] Schwefel H-P 1981 *Numerical Optimization of Computer Models* (Chichester: Wiley)

[SCHW91] Schwefel H P and Männer R (ed) 1991 *Parallel Problem Solving from Nature, Proc. 1st Workshop PPSN I (Dortmund, Germany, 1–3 October)* (Berlin: Springer)

[SCHW94] Schwehm M and Walter T 1994 Mapping and scheduling by genetic algorithms *Parallel Processing: CONPAR 94–VAPP VI, 3rd Joint Int. Conf. on Vector and Parallel Processing (Linz, Austria, September 1994) (Lecture Notes in Computer Science 854)* ed B Buchberger and J Volkert (Berlin: Springer) pp 69–74

[SCHW95] Schwehm M 1995 Optimierung der Partitionierung und Kanban-Zuordnung bei JIT-Fertigungsstrassen. In: [KUHL95] pp 11–20

[SCHW96] Schwehm M 1996 Globale Optimierung mit massiv parallelen genetischen Algorithmen *Doctoral Dissertation* Department of Computer Science, University of Erlangen-Nuremberg, Erlangen

[SEBA94] Sebald A V and Fogel L J (ed) 1994 *Proc. 3rd Ann. Conf. on Evolutionary Programming (San Diego, CA, February 24–26)* (Singapore: World Scientific)

[SEIB94] Seibulescu A 1994 Instruction scheduling on multiprocessors using a genetic algorithm. In: [KOZA94b] pp 130–9

[SEKH93] Sekharan A and Wainwright R L 1993 Manipulating subpopulations of feasible and infeasible solutions in genetic algorithms *Proc. 1993 ACM/SIGAPP Symp. on Applied Computing* pp 118–25

[SEN93] Sen S 1993 Minimal cost set covering using probabilistic methods *Proc. 1993 ACM/SIGAPP Symp. on Applied Computing* (Los Alamitos, CA: ACM-SIGAPP) pp 157–64

[SENI91] Seniw D 1991 A genetic algorithm for the travelling salesman problem *Doctoral Dissertation* University of North Carolina at Charlotte, NC

[SERE94] Seredynski F 1994 Loosely coupled distributed genetic algorithms. In: [DAVI94a] pp 514–23

[SHAW96] Shaw K J and Fleming P J 1996 Initial study of multi-objective genetic algorithms for scheduling the production of chilled ready meals *Research Report* 623, Department of Automatic Control and Systems Engineering, University of Sheffield, UK

[SHEB94] Sheble G B and Maifeld T T 1994 Unit commitment by genetic algorithm and expert-system *Electr. Power Syst. Res.* **30** 115–21

[SHEN94] Shen C-Y, Pao Y H, Yip P P C 1994 Scheduling multiple job problems with guided evolutionary simulated annealing approach *Proc. 1st IEEE Conf. on Evolutionary Computing (Orlando, FL, June 27–29 1994)* vol 2 (Piscataway, NJ: IEEE) pp 702–6

[SIEG95] Sieg G 1995 Genetische Algorithmen zum Design erfolgreicher Wettbewerbsstrategien. In: [KUHL95] pp 63–72

[SIKO92] Sikora R 1992 Learning control strategies for a chemical process: a distributed approach *IEEE Expert* **7** 35–43

[SIMP94a] Simpson A R, Dandy G C and Murphy L J 1994 Genetic algorithms compared to other techniques for pipe optimization *ASCE J. Water Resources Planning Management* **120** (4)

[SIMP94b] Simpson A R and Goldberg D E 1994 Pipeline optimisation via genetic algorithms: from theory to practice *Water Pipeline Systems* ed D S Miller (London: Mechanical Engineering Publications) pp 309–20

[SING94] Singleton A 1994 Genetic programming with C++ *Byte* **19** 171–6

[SINK95] Sinkovic V and Lovrek I 1995 Performance of genetic algorithm used for analysis of call and service processing in telecommunications. In: [PEAR95] pp 281–4

[SIRA87] Sirag D J and Weisser P T 1987 Toward a unified thermodynamic genetic operator. In: [GREF87a] pp 116–22

[SIRA95] Sirag D 1995 Forge scheduling and elevator dispatching (scheduling) *Genetic Algorithms in Production Scheduling List (E-mail List)* 21.2.1995

[SMIT85] Smith D 1985 Bin packing with adaptive search. In: [GREF85a] pp 202–6a

[SMIT87] Smith R E 1987 Diploid genetic algorithms for search in time varying environments *Proc. Ann. Southeast Regional Conf. of the ACM* (New York: ACM) pp 175–9

[SMIT92a] Smith R E and Goldberg D E 1992 Diploidy and dominance in artificial genetic search *Complex Syst.* **6** 251–85

[SMIT92b] Smith W 1992 Bin packing, personal communication

[SMIT92c] Smith S P 1992 Experiment on using genetic algorithms to learn scheduling heuristics *Proc. SPIE* vol 1707 (Bellingham, WA: Society of Photo-Optical Instrumentation Engineers) pp 378–86

[SMIT93] Smith A E and Tate D M 1993 Genetic optimization using a penalty function. In: [FORR93] pp 499–503

[SOAR94] Soares C 1994 Evolutionary computation for the job-shop scheduling problem *Technical Report* RUU-CS 52, Department of Computer Science, University of Utrecht, The Netherlands

[SONN82] Sonnenschein H 1982 A modular optimization calculation method of power station energy balance and plant efficiency *J. Eng. Power* **104** 255–9

[SOUC92] Soucek B (ed) 1992 *Dynamic, Genetic, and Chaotic Programming. The Sixth-Generation* (New York: Wiley)

[SOUL96] Soule T and Foster J A and Dickinson J 1996 Using genetic programming to approximate maximum clique. In: [KOZA96a]

[SPON89] Sponsler J L 1989 Genetic algorithms applied to the scheduling of the hubble space telescope *Telematics Informatics* **6** 181–90

[SPUT84] Sputek K 1984 Anwendung evolutorischer Suchstrategien zur Top-Down-Rechnung von Budgetierungsmodellen *Diploma Thesis* Department of Economics, Technical University of Berlin

[STAC94] Stache U 1994 Untersuchung der Eignung von Genetischen Algorithmen in der simultanen Termin- und Kapazitätsplanung *Doctoral Dissertation* Department of Computer Science, University of Dortmund, Germany

[STAN93] Stanley A and Ashlock D 1993 Iterated prisoner's dilemma with choice and refusal of partners *Artificial Life III, Proc., Santa Fe Institute Studies in the Sciences of Complexity* vol 16, ed C Langton (Reading, MA: Addison-Wesley) pp 131–76

[STAR91a] Starkweather T, Whitley D and Mathias K 1991 Optimization using distributed genetic algorithms. In: [SCHW91] pp 176–85

[STAR91b] Starkweather T, McDaniel S, Mathias K, Whitley D and Whitley C 1991 A comparison of genetic sequencing operators. In: [BELE91] pp 69–76

[STAR92] Starkweather T, Whitley D, Mathias K and McDaniel S 1992 Sequence scheduling with genetic algorithms. In: [FAND92] pp 129–48

[STAR93a] Starkweather T and Whitley D 1993 A genetic algorithm for scheduling with resource consumption. In: [FAND93] pp 567–83

[STAR93b] Starkweather T J 1993 Optimization of sequencing problems using genetic algortihms *Doctoral Dissertation* Colorado State University, CO

[STAW95] Stawowy A 1995 Scheduling, routing, layout and grouping *Genetic Algorithms in Production Scheduling List (E-mail List)* 21.4.1995

[STEF95] Stefanitsis E, Christodoulou N and Psarras J 1995 Combination of genetic algorithms and CLP in the vehicle-fleet scheduling problem. In: [PEAR95] pp 22–9

[STEN93] Stender J, Tout K and Stender P 1993 Using genetic algorithms in economic modelling: the many-agents approach. In: [ALBR93] pp 607–12

[STEN94a] Stender J 1994 Locational and sectoral modelling using genetic algorithms. In: [HILL94] pp 218–35

[STEN94b] Stender J, Tout K, Look T and Ristau P 1994 Using genetic algortihms as adaption mechanism in economic simulation models *Proc. 2nd Eur. Congress on Intelligent Techniques and Soft Computing (EUFIT'94) (Aachen, 20–23 September 1994)* vol 2 (Aachen: ELITE Foundation) pp 609–14

[STIL96] Stillger M and Spiliopoulou M 1996 Genetic programming in database query optimization. In: [KOZA96a]

[STOC93] Stockton D J and Quinn L 1993 Identifying economic order quantities using genetic algorithms *Int. J. Prod. Management* **13** 92–103

[STÖP92] Stöppler S and Bierwirth C 1992 The application of a parallel genetic algorithm to the $n/m/P/C_{\max}$ flowshop problem. In: [FAND92] pp 161–75

[STOR92] Storer R H, Wu S D and Vaccari R 1992 Local search in problem and heuristic space for job shop scheduling genetic algorithms. In: [FAND92] pp 150–60

[STOR93] Storer R H, Wu S D and Park I 1993 Genetic algorithms in problem space for sequencing problems. In: [FAND93] pp 584–97

[SUH87a] Suh J Y and Gucht D V 1987 Distributed genetic algorithms *Technical Report* 225, Computer Science Department, Indiana University

[SUH87b] Suh J Y and Gucht D V 1987 Incorporating heuristic information into genetic search. In: [GREF87a] pp 100–7

[SURR94] Surry P 1994 Gas network pipe sizing with GAs *Technical Report* EPCC-TR94–11, Edinburgh Parallel Computing Centre, Edinburgh, UK

[SURR95] Surry P D, Radcliffe N J and Boyd I D 1995 A multi-objective approach to constrained optimisation of gas supply networks: the COMOGA method. In: [FOGA95c] pp 166–80

[SYLO93] Sylogic BV 1993 Prediction of the bids of pilots for promotion to other aircraft using rule induction and a genetic algorithm; information contained in a commercial video produced by Sylogic BV, PO Box 26, 3990 DA Houten, The Netherlands

[SYSW91a] Syswerda G and Palmucci J 1991 The application of genetic algorithms to resource scheduling. In: [BELE91] pp 502–8

[SYSW91b] Syswerda G 1991 Schedule optimization using genetic algorithms. In: [DAVI91] pp 332–49

[SYSW93] Syswerda G 1993 Simulated crossover in genetic algorithms. In: [WHIT93] pp 239–55

[SZAK95] Szakal L 1995 Job shop scheduling *Genetic Algorithms in Production Scheduling List (E-mail List)* 15.3.1995

[TAM92] Tam K Y 1992 Genetic algorithms, function optimization, and facility layout design *Eur. J. Operations Res.* **63** 322–46

[TAMA92] Tamaki H and Nishikawa Y 1992 A parallel genetic algorithm based on a neighborhood model and its application to the jobshop scheduling. In: [MÄNN92] pp 573–82

[TAMA93] Tamaki H, Hasegawa Y, Kozasa J and Araki M 1993 Application of search methods to schedulung problem in plastics forming plant: a binary representation approach *Proc. 32nd IEEE Conf. on Decision and Control (32nd CDC)* (Piscataway, NJ: IEEE) pp 3845–50

[TAMA94a] Tamaki H, Mori M, Araki M, Mishima Y and Ogai H 1994 Multi-criteria optimization by genetic algorithms: a case of scheduling in hot rolling process *Technical Report* Automatic Control Engineering Group, Department of Electrical Engineering II, Kyoto University, Japan

[TAMA94b] Tamaki H, Kita H, Shimizu N, Maekawa K and Nishikawa Y 1994 A comparison study of genetic codings for the traveling salesman problem *Proc. 1st IEEE Conf. on Evolutionary Computation (ICEC'94)* (Piscataway, NJ: IEEE) pp 1–6

[TANG94] Tang A Y-C and Leung K-S 1994 A modified edge recombination operator for the traveling salesman problem. In: [DAVI94a] pp 180–8

[TANO95] Tanomaru J 1995 Staff scheduling by a genetic algorithm with heuristic operators *Proc. 1995 IEEE Int. Conf. On System, Man and Cybernetics* vols 1–5 (Piscataway, NJ: IEEE) pp 1951–6

[TATE94] Tate D M, Tunasar C and Smith A E 1994 Genetically improved presequences for Euclidean travelling salesman problems *Math. Comput. Modelling* **20** 135–43
[TATE95] Tate D M and Smith A E 1995 A genetic approach to the quadratic assignment problem *Comput. Operations Res. 22* 73–83
[TAYL95] Taylor P 1995 Modelling artificial stock markets using genetic algorithms. In: [GOON95a] ch 15
[TERA94] Terano T and Muro Z 1994 On-the-fly knowledge base refinement by a classifier system *AICOM* **7** 86–97
[TERA95] Terano T and Yoshinaga K 1995 Integrating machine learning and simulated breeding techniques to analyze the characteristics of consumer goods. In: [BIET95a] pp 211–24
[THAN91a] Thangiah S R 1991 GIDEON: A genetic algorithm system for vehicle routing with time windows *Doctoral Dissertation* North Dakota State University, Fargo, ND
[THAN91b] Thangiah S R, Nygard K E and Juell P L 1991 GIDEON: A genetic algorithm system for vehicle routing with time windows *Proc. IEEE Conf. on Artificial Intelligence Applications (Miami, FL)* (Los Alamitos, CA: IEEE Computer Society) pp 322–8
[THAN92a] Thangiah S R and Nygard K E 1992 School bus routing using genetic algorithms *Proc. SPIE Conf. on Applications of Artificial Intelligence X: Knowledge Based Systems (Orlando, FL)* (Bellingham, WA: Society of Photo-Optical Instrumentation Engineers) pp 387–9
[THAN92b] Thangiah S R and Nygard K E 1992 MICAH: A genetic algorithm system for multi-commodity transshipment problems *Proc. 8th IEEE Conf. on Artificial Intelligence for Applications (Monterey, CA, 1992)* (Piscataway, NJ: IEEE) pp 322–8
[THAN93a] Thangiah S R 1993 Vehicle routing with time windows using genetic algorithms *Technical Report* SRU-CpSc-TR-93-23, Computer Science Department, Slippery Rock University, Slippery Rock, PA
[THAN93b] Thangiah S R, Vinayagamoorthy R and Gubbi A V 1993 Vehicle routing with time deadlines using genetic and local algorithms. In: [FORR93] pp 506–13
[THAN93c] Thangiah S R and Gubbi A V 1993 Effect of genetic sectoring on vehicle routing problems with time windows *Technical Report* SRU-CpSc-TR-92–13, AI and Robotics Laboratory, Slippery Rock University, Slippery Rock, PA
[THAN95] Thangiah S R 1995 An adaptive clustering method using a geometric shape for vehicle routing problems with time windows. In: [ESHE95] pp 536–43
[THIE93] Thiel J T and Voss S 1993 Some experiences on solving multiconstraint zero-one knapsack problems with genetic algorithms *Working Paper* TH Darmstadt
[THOM86] Thompson B and Thompson B 1986 Evolving knowledge from data *Comput. Language* **11** 23–6
[TSUJ95a] Tsujimura Y, Gen M, Li Y and Kubota E 1995 An efficient method for solving fuzzy assembly-line balancing problem using genetic algorithms *Proc. 3rd Eur. Conf. on Intelligent Techniques and Soft Computing (EUFIT'95) (Aachen)* (Aachen: ELITE Foundation) pp 406–15
[TSUJ95b] Tsujimura Y, Gen M and Kubota E 1995 Solving fuzzy assembly-line balancing problem with genetic algorithms *Comput. Ind. Eng.* **29** 543–7
[TSUT93] Tsutsui S and Fujimoto Y 1993 Forking genetic algorithm with blocking and shrinking modes (fGA). In: [FORR93] pp 206–13
[TUFT93] Tufts P 1993 Parallel case evaluation for genetic programming *1993 Lectures in Complex Systems (Santa Fe Institute Studies in the Science of Complexity 10)* ed D L Stein and L Nadel (Reading/MA: Addison-Wesley)
[TUSO94a] Tuson A L 1994 The use of genetic algorithms to optimise chemical flowshops of unrestricted topology *Chemistry Part II Thesis* Oxford University, UK
[TUSO94b] Tuson A L 1994 The implementation of a genetic algorithm for the scheduling and topology optimisation of chemical flowshops *Technical Report* TRGA94-01, Oxford University
[UCHI94] Uchimura K, Sakaguchi H and Nakashima T 1994 Genetic algorithm for vehicle routing problems *Proc. 1994 Vehicle Navigation and Information Systems Conf. (Yokohama, August 31–September 2 1994)* (Piscataway, NJ: IEEE) pp 287–90
[UCKU93] Uckun S, Bagchi S, Kawamura K and Miyabe Y 1993 Managing genetic search in job shop scheduling *IEEE Expert* **8** 15–24
[ULDE91] Ulder N L J *et al* 1991 Genetic local search algorithms for the traveling salesman problem. In: [SCHW91] pp 109–16
[VALE94] Valenzuela C L and Jones A J 1994 Evolutionary divide and conquer (I): a novel genetic approach to the TSP *Evolutionary Computat.* **1** 313–33
[VÁNC91] Váncza J and Márkus A 1991 Genetic algorithms in process planning *Comput. Industry* **17** 181–94
[VASI95] Vasilakos A V, Verentziotis E A and Saltouros M F P 1995 Genetic algorithm for the Steiner tree problem: a novel approach to multimedia multicast routing *Proc. 3rd Eur. Conf. on Intelligent Techniques and Soft Computing EUFIT'95 (Aachen)* (Aachen: ELITE Foundation) pp 360–3
[VAVA95] Vavak F, Fogarty T C and Cheng P 1995 Load balancing application of the genetic algorithm in a nonstationary environment. In: [FOGA95c] pp 224–33
[VENU92] Venugopal V and Narendran T T 1992 A genetic algorithm approach to the machine-component grouping problem with multiple objectives *Comput. Ind. Eng.* **22** 469–80

[VERE95a] Vere S A 1995 Genetic classification trees. In: [BIET95a] pp 277–89
[VERE95b] Vere S A 1995 Fraud detection at Bank of America, personal communication
[VOGE95a] Voget S 1995 Epistasis in periodic programs. In: [ESHE95] pp 225–30
[VOGE95b] Voget S 1995 The integrated fixed interval timetable: optimization with genetic algorithms and fuzzy technology, paper presented at: Symposium Operations Research SOR'95, Passau, Germany, 13–15 September 1995
[VOGE95c] Voget S 1995 Aspekte genetischer Optimierungsalgorithmen. Mathematische Modellierung und Einsatz in der Fahrplanerstellung *Doctoral Dissertation* Department of Mathematics, University of Hildesheim
[VOGE95d] Voget S 1995 Speeding up a genetic algorithm for the optimization of periodic networks *Technical Report, Hildesheimer Informatik-Berichte* 21/95, Department of Mathematics, University of Hildesheim
[VOIG89] Voigt H-M 1989 *Evolution and Optimization. An Introduction to Solving Complex Problems by Replicator Networks* (Berlin: Akademie)
[VOLT93] Volta G 1993 Job shop scheduling, personal communication
[WAGN85] Wagner H 1985 Procedures for the solution of the unit commitment problem *Applied Optimization Techniques in Energy Problems* ed H J Wacker (Stuttgart: Teubner) pp 449–70
[WAH95] Wah B W, Chu L C and Aizawa A N 1995 Genetics-based learning of new heuristics—rational scheduling of experiments and generalization *IEEE Trans. Knowledge Data Eng.* **7** 763–85
[WALK94] Walker R F, Haasdijk E W and Gerrets M C 1994 Sex between models. Credit scoring using a genetic algorithm *Research Paper* CAP VOLMAC Corporation, Huis ter Heide, The Netherlands
[WALK95] Walker R F, Haasdijk E W and Gerrets M C 1995 Credit evaluation using a genetic algorithm. In: [GOON95a] ch 3
[WALT93a] Walters G A and Cembrowicz R G 1993 Optimal design of water distribution networks *Water Supply Systems, State of the Art and Future Trends* ed E Cabrera and F Martinez (Southampton, UK: Computational Mechanics Publications) pp 91–117
[WALT93b] Walters G A and Lohbeck T K 1993 Optimal layout of tree networks using genetic algorithms *Eng. Optimization* **22** 27–48
[WALT93c] Walters G A 1993 Evolutionary design for the optimal layout of tree networks *Report* 93/11, Centre for Systems and Control Engineering, University of Exeter, Exeter, UK
[WALT94] Walters G A and Savic D A 1994 Optimal design of water systems using genetic algorithms and other evolution programs *Hydraulic Engineering Software 1: Water Resources and Distribution* ed W R Blain and K L Katsifarakis (Southampton, UK: Computational Mechanics Publications) pp 19–26
[WALT95] Walters G A and Smith D K 1995 An evolutionary design algorithm for the optimal layout of tree networks *Eng. Optimization* **24** 261–81
[WALT96] Walters G A and Savic D A 1996 Recent applications of genetic algorithms to water system design *Hydraulic Engineering Software VI* ed W R Blain (Southampton, UK: Computational Mechanics Publications)
[WARR94] Warren M A 1994 Stock price time series prediction using genetic programming. In: [KOZA94b] pp 180–3
[WARW95] Warwick T and Tsang E P K 1995 Tackling car sequencing problems using a generic genetic algorithm *Evolutionary Computat.* **3** 267–98
[WATA95] Watanabe T, Hashimoto Y, Nishikawa I and Tokumaru H 1995 Line balancing using a genetic evolution model *Control Eng. Practice* **3** 69–76
[WEBE93] Weber H 1993 Entwicklung eines genetischen Verfahrens zum Bandabgleich unter Berücksichtigung praxisrelevanter Restriktionen *Diploma Thesis* Department of Economics, University of Hagen, Germany
[WEIN93] Weingarten U 1993 Ein Ansatz zur Anwendung Genetischer Algorithmen auf das Problem der Grobterminierung, paper presented at: 'Jahrestagung von DGOR und NSOR', Amsterdam
[WELL94] Weller R and Schulte J W 1994 Flexible optimisation tools for intelligent manufacturing systems *Proc. Preprints of the 2nd IFAC IFIO/IFORS Workshop IMS'94 (Vienna)* (Vienna: IFAC) pp 557–62
[WERN84] Werner F 1984 Zur Lösung spezieller Reihenfolgeprobleme *Doctoral Dissertation* Technical University Magdeburg, German Democratic Republic
[WERN88] Werner F 1988 Ein adaptives stochastisches Suchverfahren für spezielle Reihenfolgeprobleme *Ekonomicko-Matematicky Obzor* **24** 50–67
[WEZE94] van Wezel M C, Kok J N, von den Berg J and van Kampen W 1994 Genetic improvement of railway timetables. In: [DAVI94a] pp 566–75
[WHIT87] Whitley D 1987 Using reproductive evaluation to improve genetic search and heuristic discovery. In: [GREF87a] pp 108–15
[WHIT91] Whitley D, Starkweather T and Shaner D 1991 The traveling salesman and sequence scheduling: quality solutions using genetic edge recombination. In: [DAVI91] pp 350–72
[WHIT93] Whitley L D (ed) 1993 *Foundations of Genetic Algorithms 2* (San Mateo, CA: Morgan Kaufmann)
[WIEN94] Wiendahl H P and Garlichs R 1994 Decentral production scheduling of assembly systems with genetic algorithm *CIRP Ann.* **43** 389–95
[WILL96] Williams W 1996 Crew/staff scheduling with GA *Genetic Algorithms in Production Scheduling List (E-Mail List)* 29.4.1996

[WITT91] Witt U 1991 *Evolutionary Economics—An Interpretative Survey (Papers on Economics and Evolution 9104)* ed European Study Group for Evolutionary Economics (Freiburg: European Study Group for Evolutionary Economics)

[WOLP95] Wolpert D H and Macready W G 1995 No free lunch theorems for search *Technical Report* SFI-TR-95-02-010, February 1995, Santa Fe Institute, CA

[WONG95a] Wong K P and Wong Y W 1995 Thermal generator scheduling using hybrid genetic simulated annealing approach *IEE Proc.: Generation, Transmission Distribution* **142** 372–80

[WONG95b] Wong K P and Wong Y W 1995 Development of hybrid optimisation techniques based on genetic algorithms and simulated annealing. In: [YAO95a] pp 127–54

[WONG95c] Wong K P and Wong Y W 1995 Development of parallel hybrid optimisation techniques based on genetic algorithms and simulated annealing. In: [YAO95a] pp 155–65

[WONG96] Wong K P and Wong Y W 1996 Combined genetic algorithm simulated annealing fuzzy set approach to short-term generation scheduling with take-or-pay fuel contract *IEEE Trans. Power Syst.* **11** 128–35

[WREN95] Wren A and Wren D O 1995 A genetic algorithm for public transport driver scheduling *Comput. Operations Res.* **22** 101–10

[YAMA92a] Yamada T and Nakano R 1992 A genetic algorithm applicable to large-scale job-shop problems. In: [MÄNN92] pp 281–90

[YAMA92b] Yamamura M, Ono T and Kobayashi S 1992 Character-preserving genetic algorithms for travelling salesman problem *J. Japan. Soc. Artificial Intelligence* **7** 1049–59

[YANG93a] Yang C H and Nygard K E 1993 Effects of initial population in genetic search for time constrained travelling salesman problem *Proc. 21st Ann. ACM Computer Science Conf. (Indianapolis, February 16–18 1993)* ed S C Kwasny and J F Buck (New York: ACM) pp 378–83

[YANG93b] Yang J J and Korfhage R R 1993 Query optimization in information retrieval using genetic algorithms. In: [FORR93] pp 603–11

[YANG94] Yang X 1994 Evolution program for bicriteria transportation problem *Comput. Ind. Eng.* **27** 481–4

[YAO95a] Yao X (ed) 1995 *Progress in Evolutionary Computation (Lecture Notes in Artificial Intelligence 956)* (Berlin: Springer)

[YAO95b] Yao X and Darwen P J 1995 An experimental study of N-person iterated prisoner's dilemma games. In: [YAO95a] pp 90–108

[YERA94] Yeralan S and Lin C-S 1994 Genetic search and the dynamic facility layout problem *Comput. Operational Res.* **21** 955–60

[YIN91] Yin X and Germay N 1991 Investigations on solving the load flow problem by genetic algorithms *Electr. Power Syst. Res.* **22** 151–63

[YIN94] Yin X 1994 Investigation on the application of genetic algorithms to the load flow problem in electrical power systems *Doctoral Thesis* Electrotechnics and Instrumentation Laboratory, Université Catholiques de Louvain, Louvain-La-Neuve, Belgium

[YIP93] Yip P P C and Pao Y-H 1993 A new optimizer for the facility layout problem *Proc. 1993 Int. Joint Conf. on Neural Networks (Nagoya, October 25–29 1993)* (Piscataway, NJ: IEEE) pp 1573–6

[YIP94] Yip P P C and Pao Y-H 1994 A guided evolutionary simulated annealing approach to the quadratic assignment problem *IEEE Trans. Syst., Man Cybern.* **24** 1383–7

[YONG95] Yong J 1995 Production scheduling for a steel mill with 3 shifts per day *Genetic Algorithms in Production Scheduling List (E-mail List)* 18.8.1995

[YOSH94] Yoshida Y and Adachi N 1994 A diploid genetic algorithm for preserving population diversity—pseudo-meiosis GA. In: [DAVI94a] pp 36–45

[ZIMM85] Zimmermann A 1985 *Evolutionsstrategische Modelle bei einstufiger, losweiser Produktion* (Frankfurt am Main: Lang)

F1.3 Control

John R McDonnell

Abstract

This section reviews the use of evolutionary optimization techniques in automatic controller design. Specific technologies reviewed include linear control and nonlinear control using neural, fuzzy, and rule-based systems. A brief overview of evolutionary robotics is given as it pertains to both high- and low-level controller design.

F1.3.1 Introduction

This section reviews the use of evolutionary optimization techniques in automatic controller design. Because evolutionary search is amenable to a broad spectrum of optimization problems, including control system design, reformulation of the underlying search paradigm is usually unnecessary for this particular application. However, the designer still must consider the traditional issues in formulating an optimal control problem: the system model, the state and input constraints, and the performance index.

Optimal control of a system implies that an optimal control signal $\boldsymbol{u}^*$ is generated such that the dynamic system which subscribes to the state equations

$$\dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t), \boldsymbol{u}(t), t) \tag{F1.3.1}$$

follows an admissible trajectory $\boldsymbol{x}$ that minimizes a performance index such as

$$J = h(\boldsymbol{x}(t_\mathrm{f}), t_\mathrm{f}) + \int_{t_\mathrm{o}}^{t_\mathrm{f}} g(\boldsymbol{x}(t), \boldsymbol{u}(t), t)\,\mathrm{d}t \tag{F1.3.2}$$

where functions h and g are scalar.

In defense of an iterative numerical approach, as undertaken with evolutionary optimization techniques, Ogata (1995, p 567) points out that 'except for special cases, the optimal control problem may be so complicated that a computational solution has to be obtained'. It is noted that a design that minimizes a particular arbitrary performance index is not necessarily optimal in light of other performance indices, and the designer should not only strive for optimality, but should also consider the robustness of the design.

Before reviewing some applications of evolutionary computation (EC) to the controller design problem, it is worthwhile to consider some of the ramifications of using EC for on-line (adaptive) and off-line (nonadaptive) control system optimization. The idea of optimizing a set of n controllers directly driving a single plant as shown in figure F1.3.1 is usually not viable except in certain applications (e.g. robotics). It is more common to optimize the set of candidate controllers via simulation using a model of the plant. Once an acceptable controller has been evolved, it can then be implemented on the actual system, although this design approach may constrain the designer to the utilization of a nonadaptive controller.

EC can also be applied in the system identification phase for self-tuning controllers where traditional design methods are implemented for determining the control law. This approach is shown in figure F1.3.2. (See Section F1.4 for detailed discussion on the application of EC to system identification.) Adaptive control using EC suffers from the constraint that the plant under control has lower bandwidth than the corresponding rate at which control signals are generated. In addition, it is necessary to ensure that

F1.4

an acceptable control signal is generated at every sample. These issues may be best addressed using a parallel architecture for implementing adaptive control as shown in figure F1.3.3. It should be noted that it is not necessary that all of the candidate controllers be generated using evolutionary search. For example, a particular controller (or set of controllers) can be generated by more conventional means and then evaluated with the population of candidates. This hybrid approach may help to ensure that the control signals are reasonably well behaved until the evolved controllers undergo enough training generations to yield improved performance. In essence, the system shown in figure F1.3.3 could combine aspects of the approaches shown in figures F1.3.1 and F1.3.2.

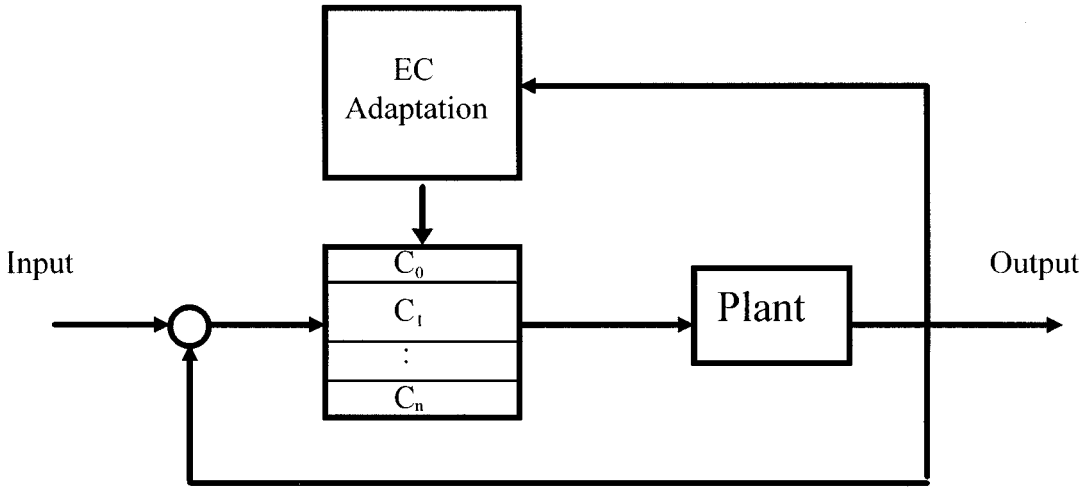


Figure F1.3.1. The output of the plant is evaluated for each of n controllers. Each controller is then modified using EC. Most implementations preclude the use of this approach at the hardware level and instead rely on off-line simulations.

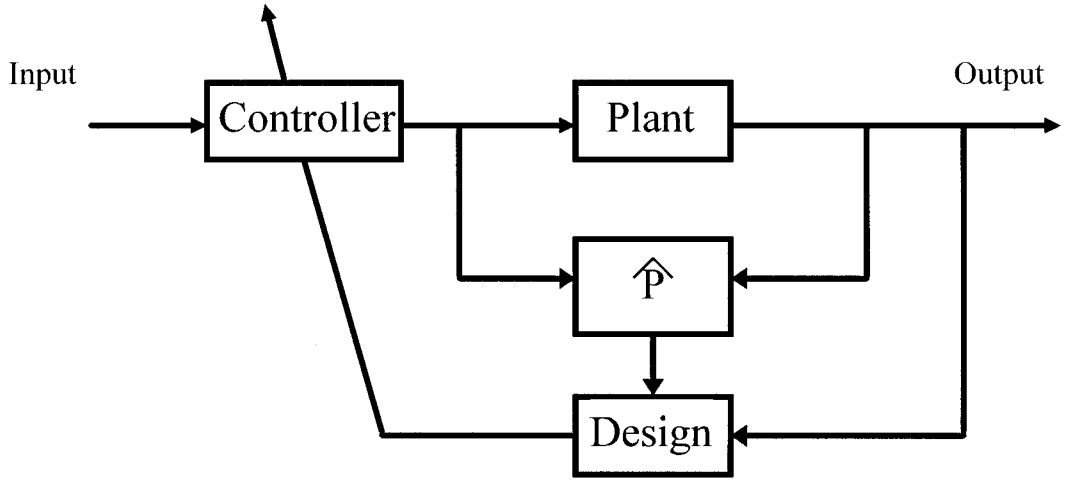


Figure F1.3.2. EC can be used to generate an estimate of the plant $\hat{P}$ (see Section F1.4 for further elaboration on this topic), and then traditional design techniques can be applied for self-tuning controller implementations.

F1.3.2 Evolutionary computation in controller design

A natural starting point for evaluating the effectiveness of evolutionary search in controller design is the design of linear controllers. Michalewicz (1992) evaluated evolutionary search to determine the optimal state feedback gains in a linear–quadratic regulator (LQR) and a variety of other problems. He showed

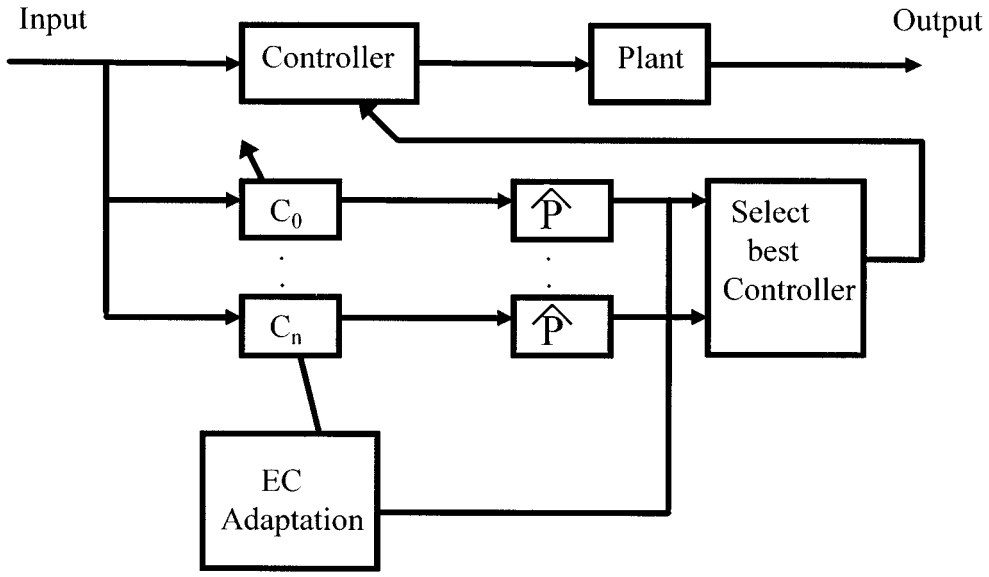


Figure F1.3.3. A parallel architecture may be used to rapidly generate and test controller designs using an estimate of the plant's dynamics.

that EC is effective in optimizing an LQR controller for different objective function weightings as well as varied system dynamics. Utilizing the control problems proposed by Michalewicz (1992), Fogel (1994) discusses an evolutionary approach that may be more robust for generating good solutions across all of the problems discussed by Michalewicz.

Lansberry *et al* (1992) used a genetic algorithm to optimize a proportional-integral (PI) controller for a linearized model of a hydrogenerator plant. Using the steady-state Riccati equation solution as a baseline, Krishnakumar and Goldberg (1992) have demonstrated that genetic algorithms can perform better than Powell's conjugate direction set method in LQR design. Saravanan (1995) has shown the effectiveness of evolutionary programming for H_∞ controller design in light of a multitude of different objectives. Although the order of the controller was assumed to be known, Saravanan correctly points out that this need not be the case as 'both the controller structure and parameters can be optimized simultaneously'. Simultaneous optimization of the model structure and its parameters has been explored by Fogel (1992, 1995) and Kristinsson and Dumont (1992) for linear and nonlinear system identification (see Section F1.4 for elaboration on this topic). Kristinsson and Dumont utilize the evolved system estimates for pole placement adaptive control. Fogel (1995) implements a nonlinear (bang–bang) controller based upon a linear system model and quadratic objective function. Both of the adaptive approaches presented by Kristinsson and Dumont (1992) and Fogel (1995) subscribe to an architecture similar to that shown in figure F1.3.2. F1.4

Because of its direct search capabilities, the application of EC in the design of linear or nonlinear controllers for nonlinear systems offers perhaps the greatest potential gain over traditional methods. For example, Varsek *et al* (1993) employ a genetic algorithm in evolving the solution to a nonlinear bang–bang controller for the classic *cart–pole system*. Krishnakumar and Goldberg (1992) demonstrate the effectiveness of a genetic algorithm for optimizing the controller of a nonlinear aircraft model subject to severe wind shear disturbance. Even though Kundu and Kawata (1996) incorporate a linear example, they demonstrate the use of a genetic algorithm to optimize nonlinear state feedback gains (in this case the states are fed back in a bilinear form). More importantly, Kundu and Kawata put forth the concept of multiple solutions generated by a genetic algorithm, any of which the control designers may select based on their preferences. D2.2

If a system can be sufficiently described with a linear model and quadratic objective function, then traditional methods of linear controller design are usually satisfactory. However, if a system contains pronounced nonlinearities that do not subscribe to linear approximation, then alternatives to traditional design approaches are necessary. *Neural network*, *fuzzy system*, and rule-based controllers are common approaches in the literature for nonlinear control. D1, D2

Evolved neural network controllers have incorporated a variety of architectures including traditional feedforward structures, recurrent nets, and the cerebellar modular articulated controller (CMAC). Wieland (1991) used genetic algorithms for training recurrent neural networks that were used to control variations on the classical cart–pole system. Wieland's results demonstrated increased performance gains with an increase in the number of generations. Saravanan and Fogel (1995) have incorporated Wieland's models in their investigations of evolving feedforward neural network controllers with only sparse (success or failure) feedback from the system. Saravanan and Fogel build upon the previous work done by Saravanan (1994a, b) who used evolutionary search in reinforcement learning control as applied to the classical cart–pole system. Pratt (1994) also has presented a technique (based on a modified cellular encoding) for evolving feedforward neural networks for nonlinear system control.

Sebald and Fogel (1990), Sebald *et al* (1991, 1992), and Sebald and Schlenzig (1994) have postulated a minimax design criterion in using EC to evolve parameters for a CMAC neural network for patient blood pressure control by drug infusion during surgery. Subsequent work by Fogel and Sebald (1995) has demonstrated the effectiveness of an adaptive approach as shown in figure F1.3.2 to the blood pressure control problem.

Fuzzy systems have been evolved in various efforts to control a multitude of nonlinear systems. Evolutionary optimization is well suited to tackle many issues in fuzzy control system design, including determining the fuzzy parameters and the shape of the membership functions. For example, Karr and Gentry (1993) have used genetic algorithms to evolve the shape of trapezoidal membership functions in on-line adaptive control of pH values. Their work subscribes to the architecture shown in figure F1.3.3 where conventional methods are used for modeling the nonlinear plant and a genetic algorithm is used to modify the controller after simulations are conducted off-line. The adaptive fuzzy logic controller of Karr and Gentry yields better response characteristics to changes in the system's dynamics than the nonadaptive fuzzy logic controller.

Karr and Gentry (1994) have also used genetic algorithms to select the most appropriate membership function (from a set consisting of exponential, sinusoidal, triangular, and symmetric trapezoidal) as well as the location and width of the membership function. Park *et al* (1994) simultaneously evolve both the membership functions and the fuzzy relation matrix using a genetic algorithm. They show better results if the system is initialized using knowledge provided by an expert in a direct current (DC) motor control application. Haffner and Sebald (1993) have also applied evolutionary search in the optimization of the membership function for heating, ventilation, and air conditioning (HVAC) control. Katai *et al* (1994) have employed genetic algorithms for optimizing the locations and widths of the different levels of the constraint and goal defuzzification interpreter. The architecture of Katai *et al* separates the observed variables (and their higher-order terms) for input into decoupled sets of fuzzy control rules that are concerned only with each subsystem's goals and constraints.

Kim and Jeon (1996) have developed a novel architecture whereby a fuzzy system preprocesses the signal to a conventional proportional-derivative (PD) controller for high-precision X–Y table control. The fuzzy system is optimized using evolutionary search and serves to eliminate the steady-state error and improve transient response performance. The fuzzy–PD hybrid controller demonstrates improved performance relative to proportional-integral-derivative (PID) control for the high-precision point-to-point positioning application outlined by Kim and Jeon.

The control engineer is not limited to neural and fuzzy architectures for nonlinear controller designs. De Jong (1980) has suggested the use of evolutionary algorithms in the formation of production rules. Grefenstette (1989) demonstrates the use of SAMUEL for generating high-level rule-based control. Other rule-based controllers have also been proposed. For example, Odetayo and McGregor (1989) use a genetic algorithm to control the classic cart–pole system where the state space has been decomposed into regions, each of which corresponds to an evolved antecedent that acts as the forcing function upon the dynamic system. Varsek *et al* (1993) extend the approach offered by Odetayo and McGregor into a three-phase learning process. The first phase consists of evolving the threshold values used in partitioning the state space into regions (although symmetry is imposed on the threshold values). The second phase consists of transforming the quantized rules into a form that is more readily interpretable. Thus, the rules are put into an *if–then*, or, equivalently, decision tree format. However this rule structure smooths the evolved partitioning and, as a result, yields poorer controller performance. This inspired a third phase of design whereby a genetic algorithm was employed to fine tune the rule threshold settings. Similar to the approach of Katai *et al* (1994), Varsek *et al* formulate their objective function to minimize the error between the desired and actual states while not violating any trajectory constraints.

Finally, Goldberg (1983, 1985a–c, 1989) has shown the effectiveness of genetic algorithms for optimizing *classifier systems* for gas pipeline operations and control where one objective is to minimize the power consumption subject to pressure constraints. B1.5.2

The emerging field of evolutionary robotics employs evolutionary algorithms in the design of high- and low-level robotic controllers. (See Section G3.7 for a case study on evolutionary robotics.) Evolutionary robotics is amenable to the generate-and-test hypothesis of EC in that candidate controllers (or control strategies) are implemented on an actual system as shown in figure F1.3.1. It is not always necessary, however, that the candidate controllers be evaluated on board the actual robot. Instead, the controllers may be evolved through computer simulations and then implemented on board the robot as demonstrated by Colombetti and Dorigo (1992), Yamauchi and Beer (1994), and others. Sometimes it is necessary to continue the evolutionary optimizations on board the robot after the controller has been evolved as discussed by Nolfi *et al* (1994) and Colombetti *et al* (1996). G3.7

Much of the work in evolutionary robotics assumes that no *a priori* knowledge is given or preengineered into the system. Thus the evolved behaviors emerge through interaction with the environment. As with other nonlinear systems, many of the controllers in evolutionary robotics utilize neural networks, although classifier systems have been extensively used by others (see e.g. Dorigo and Schnepf 1993, Colombetti and Dorigo 1992, Dorigo and Colombetti 1994, Colombetti *et al* 1996). The ability to adapt to the environment is viewed by Dorigo and Schnepf as necessary to achieve true autonomy. To facilitate learning new behaviors, Dorigo and Schnepf not only learn how to use a set of rules, but also can create new rules to incorporate into the classifier system.

Similar to results found for generic function optimization, Meeden (1996) suggests that evolutionary learning is complementary to local search methods in the context of reinforcement learning for neural network architectures which control an autonomous platform. Her recurrent neural network controller is applied to the actual system in the manner shown by figure F.1.3.1. In similar work, Floreano and Mondada (1996) show that it is possible to evolve a behavior that appears to be a combination of wall following and potential field methods, thereby avoiding local minimum problems commonly encountered with potential field methods. The issue of robustness of the controller to other environments remains an open question based on the work presented.

Research undertaken at the Naval Research Laboratory (NRL) has focused on evolving rule-based controllers for automous systems. Schultz (1994), Grefenstette (1994), and Grefenstette and Schultz (1994) describe how SAMUEL has been applied in the optimization of rule sets that are used for mobile robot navigation. Recent work by Schultz *et al* (1996) has investigated evolving complex behaviors between multiple robots. In addition to evolving robotic behaviors, NRL has also developed an automatic testing and evaluation method for autonomous vehicle controllers whereby a vehicle is subjected to an adaptively chosen set of fault scenarios (Schultz *et al* 1992).

Other EC work in robotics is of a more pragmatic nature. For example, Kim and Shim (1995) employ evolutionary search to determine gain levels of a mobile robot posture controller in an effort to achieve shortest, time-optimal paths with minimum energy expended. Their results show that EC is a viable method for determining control parameters that can be used for generating smooth trajectories. Baluja (1996) uses EC to optimize neural network weights and architectures to serve as the NAVLAB controller based upon charge-coupled device (CCD) inputs. He shows that networks trained using a genetic algorithm achieve better performance than networks trained using error backpropagation. Baluja also demonstrates the capability of evolutionary search to use a nontraditional error metric.

F1.3.3 Conclusion

In conclusion, EC offers an alternative approach to traditional methods (such as linearization or describing functions) for designing controllers, linear or nonlinear, for nonlinear systems. The computational burden imposed by evolutionary search methods should be assessed by the designer before utilizing this technique. For example, the classic cart–pole system is readily controlled using standard (and more efficient) LQR design methods on the linearized system. This does not imply that the cart–pole system is not useful in evaluating learning mechanisms. While the utilization of evolutionary search in evolving more intelligent autonomous system behaviors holds great promise, the resulting systems are far less capable than a reactive platform that incorporates path planning based on internal representations of the environment. Thus, the field of evolutionary robotics is viewed as being in its infancy with respect to more established robotic paradigms.

While the field of evolutionary controller design is fertile for additional research, one technological issue that is ripe for additional investigation is the application of evolutionary search in adaptive control. An asynchronous pooling of candidate solutions in conjunction with a parallel architecture may serve as steps toward faster generation of control signals. In addition, a less romantic but nevertheless very useful arena for further work exits in using evolutionary search in the design of programmable logic controllers (PLCs) for process control applications.

References

Baluja S 1996 Evolution of an artificial neural network based autonomous land vehicle controller *IEEE Trans. Syst. Man Cybernet.* B **SMC-26** 450–63

Colombetti M and Dorigo M 1992 Learning to control an autonomous robot by distributed genetic algorithms *From Animals to Animats 2—Proc. 2nd Int. Conf. on Simulation of Adaptive Behavior* ed J Meyer, H Roitblat and S Wilson (Cambridge, MA: MIT Press–Bradford) pp 305–12

Colombetti M, Dorigo M and Borhi G 1996 Behavior analysis and training—a methodology for behavior engineering *IEEE Trans. Syst. Man Cybernet.* B **SMC-26** 365–80

De Jong K 1980 Adaptive system design: a genetic approach *IEEE Trans. Syst. Man Cybernet.* **SMC-10** 566–74

Dorigo M and Colombetti M 1994 Robot shaping: developing autonomous agents through learning *Artificial Intell.* **71** 321–70

Dorigo M and Schnepf U 1993 Genetics-based machine learning and behavior-based robotics: a new synthesis *IEEE Trans. Syst. Man Cybernet.* **SMC-23** 141–54

Floreano D and Mondada F 1996 Evolution of homing navigation in a real mobile robot *IEEE Trans. Syst. Man Cybernet.* B **SMC-26** 396–407

Fogel D 1992 *System Identification through Simulated Evolution: a Machine Learning Approach to Modeling* (Needham, MA: Ginn)

——1994 Applying evolutionary programming to selected control problems *Comput. Math. Appl.* **27** 89–104

——1995 *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence* (Piscataway, NJ: IEEE)

Fogel D and Sebald A 1995 Steps toward controlling blood pressure during surgery using evolutionary programming *Evolutionary Programming IV: Proc 4th Ann. Conf. on Evolutionary Programming (San Diego, CA, 1995)* ed J McDonnell, R G Reynolds and D B Fogel (Cambridge, MA: MIT Press) pp 69–82

Goldberg D 1983 Computer-aided pipeline operation using genetic algorithms and rule learning *Dissertation Abstracts Int.* **44** 3174B (University Microfilms 8402282)

——1985a Controlling dynamic systems with genetic algorithms and rule learning *Proc. 4th Yale Workshop on Applications of Adaptive Systems Theory* pp 91–7

——1985b Dynamic system control using rule learning and genetic algorithms *Proc. 9th Int. Joint Conf. on Artificial Intelligence* pp 588–92

——1985c Genetic algorithms and rule learning in dynamic system control *Proc. Int. Conf. on Genetic Algorithms and Their Applications (Pittsburgh, PA, 1985)* ed J J Grefenstette (Hillsdale, NJ: Erlbaum) pp 8–15

——1989 *Genetic Algorithms in Search, Optimization and Machine Learning* (Reading, MA: Addison-Wesley)

Grefenstette J 1989 A system for learning control strategies with genetic algorithms *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, 1989)* ed J Schaffer (San Mateo, CA: Morgan Kaufmann) pp 183–90

——1994 Evolutionary algorithms in robotics algorithms *International Automation and Soft Computing: Trends in Research, Development, and Applications* ed M Jamshidi and C Nguyen (Albequerque, NM: TSI) pp 127–32

Grefenstette J and Schultz A 1994 An evolutionary approach to learning in robots *Proc. Machine Learning Workshop on Robot Learning, 11th Int. Conf. on Machine Learning, Brunswick, NJ* pp 65–72

Haffner S and Sebald A 1993 Computer-aided design of fuzzy HVAC controllers using evolutionary programming *Proc. 2nd Ann. Conf. on Evolutionary Programming (San Diego, CA, 1993)* ed D B Fogel and W Atmar (La Jolla, CA: Evolutionary Programming Society) pp 98–107

Karr C and Gentry E 1993 Fuzzy control of pH using genetic algorithms *IEEE Trans. Fuzzy Systems* **FS-1** 46–53

——1994 Control of a chaotic system using fuzzy logic *Fuzzy Control Systems* ed A Kandel and G Langholz (Boca Raton, FL: Chemical Rubber Company) pp 475–97

Katai O, Ida M, Sawaragi T, Iwai S, Khono S and Kataoka T 1994 Constraint-oriented fuzzy control schemes for cart–pole systems by goal decoupling and genetic algorithms *Fuzzy Control Systems* ed A Kandel and G Langholz (Boca Raton, FL: Chemical Rubber Company) pp 182–95

Kim J-H and Jeon J-Y 1996 Evolutionary programming-based high-precision controller design *Evolutionary Programming V: Proc. 5th Ann. Conf. on Evolutionary Programming (1996)* ed L J Fogel, P J Angeline and T Bäck (Cambridge, MA: MIT Press)

Kim J-H and Shim H-S 1995 Evolutionary programming-based optimal robust locomotion control of autonomous mobile robots *Evolutionary Programming IV: Proc. 4th Ann. Conf. on Evolutionary Programming (San Diego, CA, 1995)* ed J R McDonnell, R G Reynolds and D B Fogel (Cambridge, MA: MIT Press) pp 631–44

Krishnakumar K and Goldberg D 1992 Control system optimization using genetic algorithms *J. Guidance Control Dynam.* **15** 735–40

Kristinsson K and Dumont G 1992 System identification and control using genetic algorithms *IEEE Trans. Syst. Man Cybernet.* **SMC-22** 1033–46

Kundu S and Kawata S 1996 A GA based state feedback design method using bicriterion performance index and tournament selection *Proc. 5th Int. Conf. on Intelligent Systems* (Raleigh, NC: ISCA) pp 169–73

Lansberry J, Wozniak L and Goldberg D 1992 Optimal hydrogenerator governor tuning with a genetic algorithm *IEEE Trans. Energy Conversion* **EC-7** 623–30

Meeden L 1996 An incremental approach to developing intelligent neural network controllers for robots *IEEE Trans. Syst. Man Cybernet.* B **SMC-26** 474–84

Michalewicz Z 1992 *Genetic Algorithms + Data Structures = Evolution Programs* (New York: Springer)

Nolfi S, Florano D, Miglino O and Mondata F 1994 How to evolve autonomous robots: different approaches in evolutionary robotics *Proc. Int. Conf. on Artificial Life IV* (Cambridge, MA: MIT Press)

Odetayo M and McGregor D 1989 Genetic algorithm for inducing control rules for a dynamic system *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 177–82

Ogata K 1995 *Discrete-Time Control Systems* 2nd edn (Englewood Cliffs, NJ: Prentice-Hall)

Park D, Kandel A and Langholz G 1994 Genetic-based new fuzzy reasoning models with application to fuzzy control *IEEE Trans. Syst. Man Cybernet.* **SMC-24** 39–47

Pratt P 1994 Evolving neural networks to control unstable dynamical systems *Proc. 3rd Ann. Conf. on Evolutionary Programming (San Diego, CA, 1994)* ed A V Sebald and L J Fogel (Singapore: World Scientific) pp 191–204

Saravanan N 1994a Neurocontrol problems: an evolutionary programming approach *J. Syst. Eng.* **1** 1–12

——1994b Reinforcement learning using evolutionary programming *Proc. 3rd Ann. Conf. on Evolutionary Programming (San Diego, CA, 1994)* ed A V Sebald and L J Fogel (Singapore: World Scientific) pp 175–84

——1995 Evolutionary programming for synthesis of optimal controllers *Evolutionary Programming IV: Proc. 4th Ann. Conf. on Evolutionary Programming (San Diego, CA, 1995)* ed J R McDonnell, R G Reynolds and D B Fogel (Cambridge, MA: MIT Press) pp 645–56

Saravanan N and Fogel D 1995 Evolving neural control systems *IEEE Expert* **10** 23–7

Schultz A 1994 Learning robot behaviors using genetic algorithms *International Automation and Soft Computing: Trends in Research, Development, and Applications* ed M Jamshidi and C Nguyen (Albequerque, NM: TSI) pp 607–12

Schultz A, Grefenstette J and Adams W 1996 RoboShepherd: learning complex robotic behaviors *ISRAM '96* (Albuquerque, NM: TSI Press)

Schultz A, Grefenstette J and De Jong K 1992 Adaptive testing of controllers for autonomous vehicles *Symp. on Autonomous Underwater Vehicle Technology* (Piscataway, NJ: IEEE) pp 158–64

Sebald A and Fogel D 1990 Design of SLAYR neural networks using evolutionary programming *24th Asilomar Conf. on Signals, Systems and Computers* (San Jose, CA: Maple) pp 1020–4

Sebald A and Schlenzig J 1994 Minimax design of neural network controllers for highly uncertain plants *IEEE Trans. Neural Networks* **NN-5** 73–82

Sebald A, Schlenzig J and Fogel D 1991 Minimax design of CMAC encoded neural network controllers using evolutionary programming *25th Asilomar Conf. on Signals, Systems and Computers* (San Jose, CA: Maple) pp 551–5

——1992 Minimax design of CMAC encoded neural controllers for systems with variable time delay *Proc. 1st Ann. Conf. on Evolutionary Programming (La Jolla, CA, 1992)* ed D B Fogel and W Atmar (La Jolla, CA: Evolutionary Programming Society) pp 120–6

Varsek A, Urbancic T and Filipic B 1993 Genetic algorithms in controller design and tuning *IEEE Trans. Syst. Man Cybernet.* **SMC-23** 1330–9

Wieland A 1991 Evolving controls for unstable systems *Connectionist Models: Proc. 1990 Summer School* ed D Touretzky, J Elman, T Sejnowski and G Hinton (San Mateo, CA: Morgan Kaufmann) pp 91–102

Yamauchi B and Beer R 1994 Integrating reactive, sequential and learning behavior using dynamical neural networks *From Animals to Animats 3—Proc. 3rd Int. Conf. on Simulation of Adaptive Behavior* ed D Cliff, J Husband, J Meyer and S Wilson (Cambridge, MA: MIT Press–Bradford)

F1.4 Identification

Hitoshi Iba

Abstract

System identification techniques are applied in many fields in order to model and predict the behaviors of unknown systems given input–output data. Their practical application domains include pattern recognition, time-series prediction, Boolean function generation, and symbolic regression. Many evolutionary computation approaches have been tested in solving these problems. This section addresses brief summaries of these approaches, and compares them with alternative traditional approaches such as the group method of data handling.

F1.4.1 Introduction

The following formulation of the system identification problem was given by Zadeh (1962):

> Identification is the determination, on the basis of input and output, of a system within a specified class of systems, to which the system under test is equivalent.

System identification techniques are applied in many fields in order to predict the behaviors of unknown systems given input–output data (Astrom and Eykhoff 1971). This problem is defined formally in the following way. Assume that the single-valued output y of an unknown system behaves as a function of m input values; that is,

$$y = f(x_1, x_2, \ldots, x_m). \tag{F1.4.1}$$

Given N observations of these input–output data pairs, such as

Input				Output
x_{11}	x_{12}	...	x_{1m}	y_1
x_{21}	x_{22}	...	x_{2m}	y_2
		...		...
x_{N1}	x_{N2}	...	x_{Nm}	y_N

the system identification task is to approximate the true function f with $\hat{f}$. Once this approximate function $\hat{f}$ has been estimated, a predicted output $\hat{y}$ can be found for any input vector $(x_1, x_2, \ldots, x_m)$; that is,

$$\hat{y} = \hat{f}(x_1, x_2, \ldots, x_m). \tag{F1.4.2}$$

This $\hat{f}$ is called the 'complete form' of f. $\hat{f}$ typically has free parameters $\boldsymbol{a} = (a_1, \ldots, a_k)$, which have to be determined by a particular method. Normally, one would propose to solve the following minimization problem:

$$\sum_{i=1}^{N} (\hat{f}(a_1, \ldots, a_k \mid x_{i1}, x_{i2}, \ldots, x_{im}) - y_i)^2 \rightarrow \min. \tag{F1.4.3}$$

If $\hat{f}$ is nonlinear in $a_1, \ldots, a_k$, then the least-squares estimation of $a_1, \ldots, a_k$ is a multimodal problem. There is a clear difference between the parameter estimation with the fix model, and the system identification problem, in which the model is also optimized and searched for. However, the identification problem is converted to a certain optimization when the parameters to be estimated are defined, i.e. the model to be identified is given.

F1.4.2 Description of the application domain

An example of system identification is time-series prediction, i.e. predicting future values of a variable from its previous values. Expressed in system identification problem terms, the output $x(t)$ at time t is to be predicted from its values at earlier times $(x(t-1), x(t-2), \ldots)$:

$$x(t) = f(x(t-1), x(t-2), x(t-3), x(t-4), \ldots). \tag{F1.4.4}$$

Another example is a type of pattern recognition (or classification) problem, in which the task is to classify objects having m features $x_1, \ldots, x_m$ into one of two possible classes, i.e. 'C' and 'not C'. If an object belongs to class C, it is said to be a positive example of that class, otherwise it is a negative example. In system identification problem terms, the task is to find a (binary) function f of the m features of objects such that

$$y = f(x_1, x_2, \ldots, x_m) = \begin{cases} 0 & \text{negative example} \\ 1 & \text{positive example.} \end{cases} \tag{F1.4.5}$$

The output $y = 1$ if the object is a positive example (i.e. belongs to class C), and $y = 0$ if the object is a negative example.

The third example is a Boolean concept formation. An n-variable Boolean function is defined as a function whose ranges and domain are constrained to have zero (false) or one (true) values, i.e.

$$y = f(x_1, x_2, \ldots, x_n) = \begin{cases} 0 & \text{false value} \\ 1 & \text{true value} \end{cases} \tag{F1.4.6}$$

where

$$x_1 \in \{0, 1\} \ \wedge \ x_2 \in \{0, 1\} \ \wedge \ \ldots \ \wedge \ x_n \in \{0, 1\}. \tag{F1.4.7}$$

The goal of Boolean concept formation is to identify an unknown Boolean function, from a given set of observable input and output pairs $\{(x_{i1}, x_{i2}, \ldots, x_{in}, y_i) \in \{0, 1\}^{n+1} \mid i = 1, \ldots, N\}$, where N is the number of observations. In general, N is less than the maximum possible number of distinct n-variable Boolean functions (2^{2^n}). Since the ratio of the size of the observable data to the size of the total search space, i.e. $(N/2^{2^n})$, drastically decreases with n, effective generalizing (or inductive) ability is required for Boolean concept learning.

F1.4.3 Evolutionary computation approaches

Several researchers have applied evolutionary computation techniques to solve the system identification problems. There seems to be a natural distinction between when *genetic algorithms* (GAs) or *evolution strategies* (ESs)/*evolutionary programming* (EP) are more appropriate; that is, GAs can be applied to a problem in which decision variables are binary, whereas ESs/EP can be applied to continuous variables. However, this distinction is now disappearing. B1.2, B1.3 B1.4

Once the model to be identified is given, evolutionary algorithms (EAs) are easily applicable to solving the system identification problem, in the form of a certain parameter estimation. On the other hand, some researchers are working on the system identification itself, i.e. searching for both the model and its parameters. Typical examples can be seen in the *genetic programming* (GP) literature; some of these will be described below. B1.5.1

Kargupta and Smith (1991) extended SONN (a *simulated annealing* (SA)-based system (Tenorio and Lee 1990)) using string-based GAs and established a system called GBSON. GBSON used a GA to select nodes for a network based on an information-theoretic measure. The GBSON procedure treated the formation of each layer of a polynomial network (i.e. a network of which the layer extension led to a certain polynomial function) as a separate multimodal function optimization problem. For each layer, GBSON proceeds as follows: D3.5

(i) the first step is to generate GA structures that represent various network nodes
(ii) for each new node, the description length of the function represented is determined
(iii) a predetermined number of iterations of a GA are used to search the space of possible nodes for the current layer
(iv) after the GA execution, peak nodes are selected to form the new network layer.

The process repeats for subsequent layers until the GA converges to a layer with a single node. The resulting network is taken as a model of the input data. The preliminary results showed that simple GAs can be successful in system identification problems. However the authors also pointed out the difficulties associated with the iterative formation of complex systems with interacting elements.

The EP paradigm has also been shown to have the ability to solve system identification problems (Fogel 1991). For instance, McDonnell and Waagen (1994) experimented in evolving recurrent perceptrons for time-series modeling by means of EP. The 'perceptron' in this study referred to a recursive adaptive filter with an arbitrary output function. In their paper, a hybrid optimization scheme was proposed that embedded a single-agent stochastic search technique, the method of Solis and Wets (1981), into the EP paradigm. The proposed hybrid optimization approach was further augmented by 'blending' randomly selected parent vectors to create additional offspring. The Akaike information criterion (AIC) (see Section C4.4) was used to evaluate each recurrent perceptron structure as a candidate solution. The experimental results showed that the hybrid method can enhance EP optimization efficiency while alleviating local minimum problems associated with single-agent search techniques. The hybrid method was applied to nonlinear IIR (i.e. infinite impulse response) filters for single-step prediction tasks.

C4.4

Džeroski *et al* (1994) addressed the problem of identification of dynamical systems where a fixed model was not assumed. For this purpose, they used GP search to construct predefined building blocks (i.e. domain knowledge) that helped to generate better models, which fitted the observed behavior of dynamical systems. They applied GP to discovering a set of differential equations modeling a real-life dynamical system.

STROGANOFF (i.e. structured representation of genetic algorithms for nonlinear function fitting) is also aimed at solving system identification problems; it integrates a GP-based adaptive search of tree structures, and a local parameter tuning mechanism employing statistical search (Iba *et al* 1996). STROGANOFF was applied to several problems such as time-series prediction, pattern recognition, and zero–one optimization. The results obtained were satisfactory. The main feature was to introduce a way to modify trees, by integrating node coefficient tuning and traditional GP recombination. This approach has built a bridge from traditional GP to a more powerful search strategy (see Section G1.4 and the article by Iba *et al* (1996) for details).

G1.4

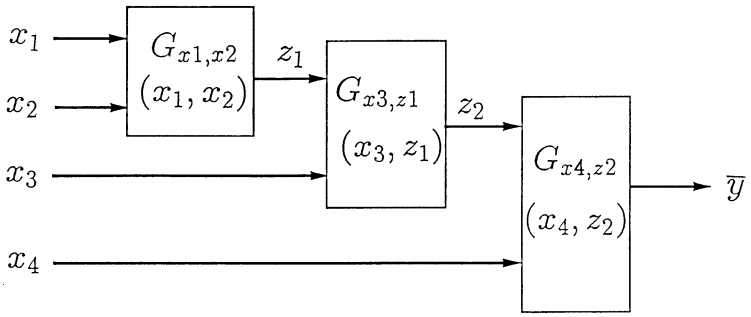


Figure F1.4.1. Feedforward network constructed by the GMDH process.

F1.4.4 Alternative approaches

System identification problems have been solved by many techniques, such as *neural networks*, *fuzzy logic*, and traditional numerical optimization methods. Among them, the group method of data handling (GMDH) and its variants are often referred to as a traditional approach (Farlow 1984). The GMDH is a multivariable analysis method which is used to solve system identification problems. This method constructs a feedforward network (as shown in figure F1.4.1) as it tries to estimate the output function $\hat{y}$. The node transfer functions (i.e. the functions G in figure F1.4.1) are quadratic polynomials of the two input variables (e.g. $G(z_1, z_2) = a_0 + a_1 z_1 + a_2 z_2 + a_3 z_1 z_2 + a_4 z_1^2 + a_5 z_2^2$) whose parameters a_i are obtained using regression techniques (Ivakhnenko 1971).

D1, D2

The GMDH uses the following algorithm to derive the 'complete form' $\hat{y}$:

Input: The observed values of the input variables (i.e. $x_1, x_2, \ldots, x_m$) and the output variable (i.e. y). The error threshold $\mathrm{Th}_{\mathrm{err}}$.
Output: The complete form, $\hat{y}$.

1 VAR $\leftarrow \{x_1, x_2, \ldots, x_m\}$;
{Initialize a set labeled VAR with the input variables.}
2 $z_1 \leftarrow$ random(VAR);
3 $z_2 \leftarrow$ random(VAR);
{Select any two elements z_1 and z_2 from the set VAR.}
4 $z \leftarrow G_{z1,z2}(z_1, z_2)$;
{Form an expression G_{z_1,z_2} which approximates the output y (in terms of z_1 and z_2) with least error using multiple-regression techniques.
Regard this function as a new variable z.}
5 **if** error$(z) \leq \mathrm{Th}_{\mathrm{err}}$
then return z;
{If z approximates y better than some criterion, set the 'complete form' (i.e. $\hat{y}$) as z and terminate.}
6 VAR $\leftarrow$ VAR $\cup \{z\}$;
7 **goto Step2**;

The important decisions to be taken when running the GMDH algorithm are as follows:

(i) the form of the subexpression for G;
(ii) the selection of the variables $\{z_1, z_2\}$ in Step2 and Step3; and
(iii) the termination criterion in Step5.

The original GMDH algorithm (Ivakhnenko 1971) used several heuristics, called 'regularizations', to generate candidates for the G expressions and for the selection of variables (i.e. z_i). However, the heuristic nature of the original GMDH led to such weaknesses as combinatorial explosiveness and becoming trapped in local minima. STROGANOFF and GBSON have extended this GMDH process with a GA- or GP-based adaptive method in order to reduce the above computational burden. The remarkable features of the EA-based approach to the identification problem are summarized as follows:

(i) As can be seen in STROGANOFF (see Section G1.4), it is possible to search for both the model and its parameters simultaneously. G1.4
(ii) The model selection criterion, such as MDL or AIC, can be introduced to evolve a desirable structure. This is to evaluate the tradeoff between the error and the model complexity (see Section C4.4 for more details). C4.4
(iii) The premature convergence to local optima could be avoided by using adaptive search. Thus, STROGANOFF and GBSON are regarded as the integration of GMDH-based local search and GP/GA-based global search (see Section G1.4).

References

Astrom K J and Eykhoff P 1971 System identification, a survey *Automatica* **7** 123–62

Džeroski S Todorovski L and Petrovski I 1994 Dynamical system identification with machine learning *Open Syst. Information Dynam.* **3** 1–23

Farlow S J (ed) 1984 *Self-Organizing Methods in Modeling, GMDH Type Algorithms* (New York: Dekker)

Fogel D B 1991 *System Identification through Simulated Evolution: a Machine Learning Approach to Modeling* (Needham, MA: Ginn)

Iba H, deGaris H and Sato T 1996 A numerical approach to genetic programming for system identification *Evolut. Comput.* **3**

Ivakhnenko A G 1971 Polynomial theory of complex systems *IEEE Trans. Syst. Man Cybernet.* **SMC-1** 367–78

Kargupta H and Smith R E 1991 System identification with evolving polynomial networks *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, July 1991)* ed R K Belew and L B Booker (San Mateo, CA: Morgan Kaufmann) pp 370–6

McDonnell J R and Waagen D 1994 Evolving recurrent perceptrons for time-series modeling *IEEE Trans. Neural Networks* **NN-5** 24–38

Solis F J and Wets J B 1981 Minimization by random search techniques *Math. Operat. Res.* **6** 19–50

Tenorio M F and Lee W 1990 Self-organizing network for optimum supervised learning *IEEE Trans. Neural Networks* **NN-1** 100–9

Zadeh L A 1962 From circuit theory to system theory *Proc. IRE 50*

F1.5 Scheduling

Ralf Bruns

Abstract

Over the past few years, a continually increasing number of research efforts have investigated the application of evolutionary computation techniques for the solution of scheduling problems. Scheduling problems can pose extremely complex combinatorial optimization problems. The necessity to satisfy various kinds of constraint makes the scheduling task even more difficult. The major portion of this section is devoted to a comprehensive overview of evolutionary computation research on scheduling. The approaches are discussed with respect to the kind of solution representation used. Evolutionary computation seems to be especially well suited for scheduling tasks where a high-quality schedule must be generated in a limited time.

F1.5.1 Introduction

Scheduling is an economically very important yet computationally extremely difficult task. Scheduling problems can be identified in several different application areas, and very diverse items are the subject of scheduling, such as *production operations in manufacturing industry*, *computer processes in operating systems*, truck movements in transportation, *aircraft crews*, and refurbishment of space shuttles. The great practical importance makes scheduling a permanently active area of research. G9.3, G1.1 G9.4

In recent years, several efforts have sought to investigate and exploit the application of evolutionary computation techniques to various scheduling problems. The main difficulty encountered is that of specifying an appropriate representation of feasible schedules.

The major portion of this section is devoted to a comprehensive overview of evolutionary computation research on scheduling problems. The different evolutionary algorithms are reviewed with respect to their problem representations and their advantages and disadvantages are discussed. Furthermore, the scheduling domain is introduced in some detail and alternative approaches to scheduling are presented. The section concludes with a discussion of the prospects of evolutionary computation for scheduling.

F1.5.2 Description of scheduling domain

Scheduling problems are prominent combinatorial optimization problems. The task of scheduling is the allocation of jobs over time to limited resources, where a number of objectives should be optimized and several constraints must be satisfied. A job is completed by a predefined sequence of operations. The result of scheduling is a schedule showing the assignment of start times and resources to each operation. The assignment effects the optimality of a schedule with respect to criteria such as cost or throughput.

The flow shop scheduling problem is a restricted scheduling problem, where the machine sequences are identical for all jobs. As a consequence, a schedule is determined by the order of the jobs introduced into the flow shop. Another restricted scheduling problem is the *job shop scheduling problem*. In a job shop, each job requires every machine exactly once and all jobs may start at the first time slot. The objective is to minimize the makespan. G1.2.3

Practical scheduling problems usually possess a more complex problem structure, because several different constraints may be relevant in realistic problem situations, such as alternative process plans for the manufacturing of a product, specialized production structures, and so forth.

The structure of a general scheduling problem can be described by the quadruple (R, P, J, C) as follows:

- R is a set of *resources*, for example, machines or personnel, with different functional capabilities.
- P is a set of *products*. Each product can be manufactured in different procedures (recipes), called *process plans*. Each process plan consists of a sequence of *operations*. A set of alternative resources exists for the processing of each operation, with a given duration.
- J is a set of *jobs* which are to be scheduled subject to several constraints defined in C. For each job the product to be produced, the ready time, and the due date are given.
- C is a set of hard *constraints*, for example, precedence relationships or capacity restrictions, that must be satisfied. Usually various kinds of different application-specific constraint exist.

The quality of a schedule is measured by means of an objective function, which assigns a numerical value to a schedule. Different objective criteria can be identified in scheduling, for example, makespan, tardiness, inventory cost, and transportation time. Usually not only a single criterion is relevant for a particular application but a combination of (sometimes conflicting) organizational objectives, for example, minimize work-in-process time and maximize resource utilization. Thus, scheduling is typically a *multicriterion optimization problem*. In summary, the goal of scheduling is the construction of a complete and feasible schedule, which minimizes/maximizes the chosen objective function. C4.5, F1.9

Apart from some theoretical cases of little practical importance, the determination of an optimal solution to a scheduling problem belongs to the class of NP-hard problems, which means that no deterministic algorithm is known yet for solving the problem in polynomial time. Practice has proven that scheduling is also an extremely difficult task for human experts. In addition to the combinatorial complexity, when dealing with real-world scheduling problems the necessity to regard different kinds of specific constraint imposed by numerous details of a particular application, for example, physical constraints on resource capabilities and utilization requirements, operating preferences, and technical constraints describing manufacturing procedures, make the scheduling task even more difficult. Scheduling can be specified as a complex constraint satisfaction problem.

F1.5.3 Evolutionary computation approaches for scheduling

Research on the application of evolutionary computation to scheduling problems is relatively recent. Almost all previous published attempts are based on the *genetic algorithm* (GA) model. Therefore, if not explicitly mentioned otherwise, the algorithms presented in this section are (variants of) GAs. The approaches are discussed with respect to the kind of representation of solutions used. The representational scheme can be either indirect or direct (Bagchi *et al* 1991). Furthermore, two different kinds of indirect representation can be distinguished: domain-independent and problem-specific ones. B1.2

F1.5.3.1 Approaches based on domain-independent indirect representation

Most evolutionary algorithms for scheduling use an indirect representation of solutions, that is, the algorithm works on a population of encoded solutions. Since the representation does not directly represent a schedule, a transition from representation to a legal schedule has to be performed by a schedule builder prior to evaluation. The schedule builder guarantees the feasibility of the schedules. Since it has to search for information not provided by the individual, its activity relies on the amount of information included in the representation—the more information is incorporated in the representation, the less search has to be performed by the schedule builder, and vice versa.

A domain-independent indirect representation scheme contains no auxiliary information regarding the particular scheduling problem. The evolutionary algorithm performs blind reproduction of encoded solutions by applying conventional operators. The domain knowledge remains separated within the evaluation procedure to determine the fitness.

Binary representation. A solution to a scheduling problem is represented by a *bit string* and the representation is subject to conventional operators such as *one-point crossover*. The approaches differ in the meaning of each bit. Cleveland and Smith (1989) scheduled flow shop release times. The release time of each job is represented as a binary integer. These times are concatenated into one long bitstring which is taken as a solution. In the article by Nakano and Yamada (1991) each bit determines which one C1.2, C3.3.1

of two jobs should be executed first on a particular machine. Since most offspring solutions produced by conventional crossover are illegal, a *repair algorithm* is employed to generate a legal individual as similar as possible to the illegal one. Tamaki and Nishikawa (1992) represented a schedule by means of a disjunctive graph. For every pair of disjunctive arcs, one bit is allotted to indicate which arc is chosen. The arcs determine the order of competing operations on one machine. Fox and McMahon (1991) designed a Boolean matrix representation of a sequence of operations that encapsulates all information about the sequence. New operators were developed to preserve the necessary properties of the sequence. C5.4

Sequence of jobs representation. The list of all jobs to be scheduled is represented as an individual. The ordering of the list represents the scheduling priority of the jobs. Thus, the scheduling problem is regarded as a sequencing problem, like the *traveling salesman problem* (TSP). The evolutionary algorithm iteratively generates new permutations of the list of jobs. For each individual the schedule builder generates the corresponding schedule according to the sequence of the jobs—the first job on the list is scheduled first, then the second one, and so on. G9.5

Several approaches applied this problem representation—in most cases for flow shop problems (Biegel and Davern 1990, Bierwirth 1993, Bruns 1992, Cleveland and Smith 1989, Lawton 1992, Muller *et al* 1993, Starkweather *et al* 1991, Stöppler and Bierwirth 1992, Syswerda 1991, Syswerda and Palmucci 1991, Whitley *et al* 1989, 1991). Several different sequencing operators were developed and the schedule builders used range from fairly simple ones to complex knowledge-based systems. *Evolution strategies* based on the sequence of jobs representation have been developed as well by Ablay (1979) and Schöneburg and Heinzmann (1992). B1.3

Sequence of operations representation. In Fang *et al* (1993) and Morikawa *et al* (1992) the representation scheme contains for each operation the number of the corresponding job. The approach proceeds in a manner similar to the aforementioned sequencing representation. The schedule builder treats the operations which belong to the same job according to the precedence relation, for example, let (3 1 3 2) be an individual, then the first operation of job3 is scheduled first, then the first one of job1, then the second one of job3, and so forth.

List of processor numbers representation (Kidwell 1993). A solution of the problem to distribute tasks over a multiprocessor system is represented by the list of processor numbers. In a first step the list of tasks is sorted. Then a GA generates sequences of processor numbers and a schedule builder assigns a task on the list to the processor whose number appears at the corresponding position in the sequence.

Random key representation (Bean 1994). A schedule is encoded by random numbers. These values are used as sort keys to decode the solution. Each job is associated to a position in the representation. A schedule is derived by sorting the random keys, and the jobs are sequenced in the order of the sort, for example, the individual (0.45, 0.36, 0.79, 0.81) would represent the job sequence job2–job1–job3–job4. Since operators are executed on the random keys, all offspring are guaranteed to be feasible.

F1.5.3.2 Approaches based on problem-specific indirect representation

In a problem-specific indirect representation scheme, knowledge of the investigated scheduling problem is explicitly represented in the individuals. In order to work on the resultant expanded representation, new domain-dependent recombination operators have to be designed. The domain knowledge is spread over the representation, the operators, and the evaluation procedure.

Sequence of job–process plan representation and sequence of job–process plan–machines representation (Bagchi et al 1991, Uckun et al 1993). In addition to the sequence of jobs, the selected process plan for each job is included into the representation. Each individual is a list of job–process plan items. The schedule builder schedules the jobs one by one according to their sequence using the specified process plans. Additionally, the set of machines to be used is incorporated into a second expanded representation. Here, each individual is a list of job–process plan–machines items. This information comprises the entire search space because the investigated problem contains neither ready times nor due dates. Domain-independent operators are employed for the generation of permutations of the list of items. Moreover, a

problem-specific crossover exchanges process plans (or machines) and a problem-specific mutation selects alternative process plans (or alternative sets of machines).

Preference list representation. The first GA-based approach for scheduling was developed by Davis (1985). It is based on a time-dependent preference list representation where a schedule is specified by a preference list for each workstation at each time interval. A preference list is a list of jobs, plus the elements 'wait' and 'idle'. It is interpreted as showing which job the workstation should prefer to execute at a given time or whether it should wait or stand idle. The introduced crossover operator exchanges preference lists between solutions, the scramble operator rearranges the contents of a preference list, and the run–idle operator can insert idle times for machines.

Cleveland and Smith (1989) and Hou and Li (1991) adjusted the approach to schedule flow shop releases and automated guided vehicles in flexible manufacturing systems, respectively. Falkenauer and Bouffouix (1991) designed a modified order crossover, which operates independently on each of the preference lists.

F1.5.3.3 Approaches based on direct representation

In a direct problem representation the complete and feasible schedule itself is used as an individual. All information relevant to uniquely describe a particular schedule is included in the representation. The evolutionary algorithm is the only method that carries out search since the represented information comprises the entire search space. A schedule builder is no longer necessary. The extended representation necessitates the definition of new recombination operators since the familiar domain-independent operators would hardly ever produce consistent offspring.

List of job–machine–start time representation. In the article by Kanet and Sridharan (1991) a schedule is represented by the list of jobs where each job has an assigned machine and a scheduled start time. Since each job is completed by exactly one operation, each individual represents a complete schedule. Offspring are created by selecting start times from several parent schedules and adjusting them to form a legal schedule.

Filipic (1992) developed a similar representation also for a single-operation jobs problem. In this representation the value at the ith position denotes the setup time of the ith job. A unique operator was devised that performs the role of both crossover and mutation.

List of operation completion times representation (Yamada and Nakano 1992). Each individual represents a schedule directly by the list of completion times for each operation. This representation is unique since the approach deals with job shop problems. The crossover can be viewed as a simple scheduling algorithm which produces a new schedule based on the idea of the active schedule generation of Giffler and Thompson. A further improvement of performance was achieved by applying a specific genetic algorithm model (Davidor *et al* 1993).

List of operation–process plan–machine–production interval representation (Bruns 1993a, b). The addressed scheduling problem comprises several additional features such as alternative process plans, alternative machines, release/due dates, and further domain-specific constraints. The scheme directly represents a feasible schedule by the list of operation–process plan–machine–production interval items. Complex knowledge-augmented crossover and mutation operators were designed to work directly on the schedules. To guarantee that all constraints remain satisfied during reproduction the operators have the functionality of knowledge-based scheduling algorithms (Appelrath and Bruns 1994).

F1.5.3.4 Other approaches

Some interesting approaches follow that do not fit into the classification used above.

Learning techniques.

- Hilliard *et al* (1988) applied *classifier systems* to discover general rules for job shop scheduling. The scheduling rules determine where to place which job in a queue. The learning objective is to learn to order job queues. B1.5.2
- Dorndorf and Pesch (1992) conducted a probabilistic learning approach, where each item in an individual represents one rule of a set of decision rules. The item at the ith position says that a conflict in the ith iteration of a heuristic scheduling algorithm should be resolved using the ith decision rule. The algorithm searches for the best sequence of decision rules for selecting operations to guide the search of a heuristic scheduling algorithm.

Joint lot sizing and sequencing. The two related problems of determining lot sizes and job sequences are approached concurrently.

- Lee *et al* (1993) first splits the original lot sizes into certain small lot units. The start population contains permutations of these small lot units. New permutations are generated using the edge recombination operator. If some types of job cluster, then the clusters are coalesced to a single lot. The representation structure evolves gradually.
- The evolution strategy developed by Zimmermann (1985) uses a representation where each digit specifies a job type and the number of consecutive equal digits specifies the lot size. The lot sizes are mutated by simultaneous duplication and deletion of digits. An inversion operator alters the production sequence.

Other methods.

- A parallel approach for integrated planning and scheduling is proposed by Husbands *et al* (1990) (see also Husbands and Mill 1991 and Husbands 1993). Separate populations of process plans evolve independently and are combined through a scheduler that builds schedules and returns fitness values.
- Paredis (1992, 1993) proposed a general method for *constraint handling* in GAs and applied it to job shop scheduling. The members of the population represent search states (partial schedules) from which solutions can be found by constraint-based search. C5

F1.5.3.5 Comparison between different evolutionary computation approaches

Only a few empirical evaluations have been reported that compare different evolutionary computation solutions for scheduling.

The most intensive evaluations were performed for the sequence of jobs representation. Fox and McMahon (1991), Starkweather *et al* (1991) and Syswerda (1991) compared different sequencing operators. Interestingly, it could be observed that the operators performed very differently on scheduling and TSP—the operators that performed well on scheduling performed rather poorly on TSP, and vice versa (Michalewicz 1992, chapter 11.2).

Several GAs were developed for the $n \times m$ (minimum-makespan) job shop problem, where n denotes the number of jobs and m the number of machines. Experiments were conducted using the famous 10×10 and 20×5 benchmark problems introduced by Muth and Thompson (1963). The reported results are shown in table F1.5.1. The listed average makespans refer to the mean result over a certain number of trials. In the article by Nakano and Yamada (1991) only the best makespan achieved was published.

Table F1.5.1. Muth–Thompson benchmark: average makespan of GA approaches.

Paper	Representation	10×10	20×5
Nakano and Yamada (1991)	binary	965 (best)	1215 (best)
Yamada and Nakano (1992)	direct	975	1236
Davidor *et al* (1993)	direct—parallel	963	1213
Fang *et al* (1993)	sequence of operations	977	1215
Optimal makespan		930	1165

The different GAs were able to obtain very good results (close to the optimum) for these difficult scheduling problems. The best performance was achieved with a direct problem representation by Davidor *et al* (1993) and a sequence of operations representation by Fang *et al* (1993), while the binary GA performed the worst.

Comparisons between domain-independent and problem-specific evolutionary algorithms were conducted as well. Experiments were run by Bagchi *et al* (1991) in order to compare the sequence of jobs representation with their problem-specific ones. The empirical evaluation of all three representations showed that the more problem-specific information included the better the schedules obtained. Bagchi *et al* (1991) drew the conclusion that all information that pertains to the optimization problem should be represented in the individual. In the article by Bruns (1993a) a complex direct representation with knowledge-augmented operators was compared with a domain-independent sequence of jobs representation. The observations gained in extensive experiments were in accordance with the results obtained by Bagchi *et al* (1991), namely that the knowledge-augmented genetic algorithm indeed generated much better schedules than the domain-independent one.

F1.5.3.6 Advantages and disadvantages

The specification of a suitable representation of a schedule is of decisive importance for the performance of an evolutionary algorithm. The representations used so far vary from simple strings to complex data structures. As a consequence, the employed operators vary as well from rather simple ones to complex knowledge-based algorithms.

The domain-independent representations seem to be especially appropriate for scheduling problems with only few constraints and a fairly simple problem structure. In particular flow shop problems have been approached very successfully with a sequence of tasks, either jobs or operations, representation.

However, in addressing more complex scheduling problems, these simple representation schemes seem to have the disadvantage that the evolutionary algorithm is restricted to perform a search only on a part of the complete search space. The rest of the search task has to be accomplished by the schedule builder. Several attempts have been made to overcome these limitations by incorporating problem-specific knowledge in the representation, thus achieving a significant improvement of performance.

Consequently, the specification of the best-suited representation structure is highly dependent on the investigated scheduling problem.

F1.5.3.7 State of the art

Scheduling problems have been the subject of considerable research efforts of the evolutionary computation community. Several genetic algorithms and a couple of evolution strategies have been developed so far for application areas such as scheduling of production processes in chemical and manufacturing industry, training exercises in a naval laboratory, and shipping of beer production. These algorithms differ essentially in the specification of a suitable problem representation and in the employed recombination operators.

The problems investigated so far are mainly rather simple versions of scheduling problems, for example, flow shop problems, job shop problems, and one-machine problems. These problems have been approached successfully by attempts which in most cases made use of domain-independent indirect representation structures. Only a few attempts have been reported where real-world scheduling problems were addressed, which are usually much more complex due to the necessity to consider additional kinds of constraint. Motivated by the problem of how to handle different kinds of domain-specific constraint, some evolutionary algorithms were specifically tailored to scheduling by the integration of problem-specific knowledge in the representation and the operators.

The major difficulty in applying evolutionary techniques to scheduling is still the suitable representation and handling of the various constraints encountered in scheduling problems.

F1.5.4 Alternative approaches

Scheduling problems have been investigated intensively in the areas of operations research and artificial intelligence. Traditionally, scheduling research has focused on methods for obtaining optimal solutions to simplified problems, for example, with integer programming or branch-and-bound algorithms. In order to determine an optimal solution, different restrictions were imposed on the problem domain, for example,

on the number of jobs or machines, which made the application of the results to more complex problems very difficult or even impossible.

Due to the difficulty of the scheduling domain, in many real-world scheduling environments the objective is the determination of a feasible schedule in reasonable time, which need not necessarily be an optimal schedule, but should, of course, be as good as possible. Heuristic algorithms have been designed to efficiently generate feasible schedules. However, the quality of the schedules found is often not satisfactory. Operations research methods for scheduling are described in more detail by Blazewicz *et al* (1994).

In recent years, an increasing interest in the use of artificial intelligence technologies in the scheduling area could be observed. Several knowledge-based scheduling systems have emerged using different paradigms, for example, rule-based approaches, constrained directed search, *fuzzy logic*, or multiagent approaches. In addition, the important practical problem of adapting an existing schedule due to actual events in the dynamic scheduling environment (reactive scheduling) has been approached with knowledge-based techniques. A comprehensive overview of knowledge-based research for scheduling can be found in the publication of Smith (1992) and Zweben and Fox (1994). D2

Moreover, other probabilistic search algorithms such as simulated annealing or threshold accepting have been applied to scheduling as well. An overview and comparison of different probabilistic search methods for scheduling is given by Dorn (1995).

F1.5.4.1 Comparison with evolutionary computation approaches

The major advantage of evolutionary computation is that the search remains tractable in terms of computing time and resources, even for complex scheduling problems of realistic size. This is in contrast to algorithms that guarantee to find the optimum, which are only applicable to very restricted scheduling problems. Hence, evolutionary computation possesses the potential to generate high-quality schedules with acceptable computing effort for many real-life scheduling problems. Further advantages of evolutionary algorithms are the possibility to interrupt the search at any time, so that a schedule is always immediately available if necessary and to cope with multicriterion objective functions.

As opposed to knowledge-based systems it is not possible for a human expert to verify the plausibility of the problem-solving process of an evolutionary algorithm. This is a significant drawback for the acceptance of evolutionary approaches in practical applications since the human expert still has the ultimate responsibility for all decisions. Besides, many real-life scheduling environments require a real-time decision-making process to keep the manufacturing process moving, for example, when unforeseen disturbances occur at a high frequency. The immediate reaction is much more important in such situations than optimization. Heuristic or knowledge-based algorithms seem to be more appropriate for real-time scheduling because they usually need much less computing time.

Evolutionary computation techniques cannot replace all the developed scheduling methods, but they have the potential to become a complementary part in hybrid scheduling systems. They seem to be especially well suited for scheduling tasks where a high-quality schedule must be generated in limited time, without real-time requirements or the guarantee to reach the global optimum. Yet, compared to conventional scheduling approaches, evolutionary techniques and other probabilistic search algorithms, in particular simulated annealing, share similar advantages/disadvantages and, consequently, do have to compete for the same scheduling tasks.

References

Ablay P 1979 *Optimieren mit Evolutionsstrategien—Reihenfolgeprobleme, nichtlineare und ganzzahlige Optimierung* Dissertation, Wirtschafts- und Sozialwissenschaftliche Fakultät, Universität Heidelberg

Appelrath H-J and Bruns R 1994 Genetische Algorithmen zur Lösung von Ablaufplanungsproblemen *Fuzzy Logik—Theorie und Praxis* ed B Reusch (Berlin: Springer) pp 25–33

Bagchi S, Uckum S, Miyabe Y and Kawamura K 1991 Exploring problem-specific recombination operators for job shop scheduling *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, July 1991)* ed R K Belew and L B Booker (San Mateo, CA: Morgan Kaufmann) pp 10–7

Bean J 1994 Genetics and random keys for sequencing and optimization *ORSA J. Comput.* **6** 154–60

Biegel J E and Davern J J 1990 Genetic algorithms and job shop scheduling *Comput. Indust. Eng.* **19** 81–91

Bierwirth C 1993 *Flowshop Scheduling mit parallelen Genetischen Algorithmen* (Deutscher Universitätsverlag)

Blazewicz J, Ecker K H, Schmidt G and Weglarz J 1994 *Scheduling in Computer and Manufacturing Systems* 2nd edn (Berlin: Springer)

Bruns R 1992 Incorporation of a knowledge-based scheduling system into a genetic algorithm *GI-Jahrestagung—Information als Produktionsfaktor* ed W Görke, H Rininsland and M Syrbe (Berlin: Springer) pp 547–53

——1993a Direct chromosome representation and advanced genetic operators for production scheduling *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 352–9

——1993b Knowledge-augmented genetic algorithm for production scheduling *Workshop Notes IJCAI-93 Workshop on Knowledge-Based Production Planning, Scheduling, and Control (Chambery, 1993)* ed N Sadeh pp 49–58

Cleveland G A and Smith S F 1989 Using genetic algorithms to schedule flow shop releases *Proc. 3rd Int. Conf. on Genetic Algorithms* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 160–9

Davidor Y, Yamada T and Nakano R 1993 The ECOlogical Framework II: improving GA performance at virtually zero cost *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 171–6

Davis L 1985 Job shop scheduling with genetic algorithms *Proc. Int. Conf. on Genetic Algorithms and their Applications (Pittsburgh, PA)* ed J J Grefenstette (Hillsdale, NJ: Lawrence Erlbaum Associates) pp 136–40

Dorn J 1995 Iterative improvement methods for knowledge-based scheduling *AI Commun.* **8** 20–34

Dorndorf U and Pesch E 1992 *Evolution Based Learning in a Job Shop Scheduling Environment* Research Memorandum RM 92-019, Rijksuniversiteit Limburg

Falkenauer E and Bouffouix S 1991 A genetic algorithm for job shop *Proc. IEEE Int. Conf. on Robotics and Automation (Sacramento, CA)* (Piscataway, NJ: IEEE) pp 824–9

Fang H-L, Ross P and Corne D 1993 A promising genetic algorithm approach to job-shop scheduling, rescheduling, and open-shop scheduling problems *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 375–82

Filipic B 1992 Enhancing genetic search to schedule a production unit *Proc. 10th Eur. Conf. on Artificial Intelligence* ed B Neumann (Chichester: Wiley) pp 603–7

Fox B R and McMahon M B 1991 Genetic operators for sequencing problems *Foundations of Genetic Algorithms* ed G J E Rawlings pp 284–300

Hilliard M R, Liepins G E and Palmer M 1988 Machine learning applications to job shop scheduling *Proc. AAAI-SIGMAN Workshop on Production Planning and Scheduling (St Paul)*

Hou E S H and Li H-Y 1991 Task scheduling for flexible manufacturing systems based on genetic algorithms *Proc. IEEE Int. Conf. on Systems, Man and Cybernetics* (Piscataway, NJ: IEEE) pp 397–402

Husbands P 1993 An ecosystem model for integrated production planning *Int. J. Comput. Integrated Manufacturing* **6** 74–86

Husbands P and Mill F 1991 Simulated co-evolution as the mechanism for emergent planning and scheduling *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, July 1991)* ed R K Belew and L B Booker (San Mateo, CA: Morgan Kaufmann) pp 264–70

Husbands P, Mill F and Warrington S 1990 Genetic algorithms, production plan optimisation and scheduling *Proc. 1st Workshop on Parallel Problem Solving from Nature (Dortmund, 1990)* ed H-P Schwefel and R Männer (Berlin: Springer) pp 80–4

Kanet J J and Sridharan V 1991 PROGENITOR: a genetic algorithm for production scheduling *Wirtschaftsinformatik* **33** 332–6

Kidwell M D 1993 Using genetic algorithms to schedule distributed tasks on a bus-based system *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 368–74

Lawton G 1992 Genetic algorithms for schedule optimization *AI Expert* **5** 23–7

Lee I, Sikora R and Shaw M J 1993 Joint lot sizing and sequencing with genetic algorithms for scheduling: evolving the chromosome structure *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 383–9

Michalewicz Z 1992 *Genetic Algorithms + Data Structures = Evolution Programs* (Berlin: Springer)

Morikawa K, Furuhashi T and Uchikawa Y 1992 Single populated genetic algorithm and its application to jobshop scheduling *Proc. Int. Conf. on Industrial Electronics, Control, and Instrumentation* (Piscataway, NJ: IEEE) pp 1014–8

Muller C, Magill E H, Prosser P and Smith D G 1993 Distributed genetic algorithms for resource allocation *Scheduling of Production Processes* ed J Dorn and K A Froeschl (Chichester: Ellis Horwood) pp 70–8

Muth J F and Thompson G L 1963 *Industrial Scheduling* (Englewood Cliffs, NJ: Prentice-Hall)

Nakano R and Yamada T 1991 Conventional genetic algorithm for job shop problems *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, July 1991)* ed R K Belew and L B Booker (San Mateo, CA: Morgan Kaufmann) pp 474–9

Paredis J 1992 Exploiting constraints as background knowledge for genetic algorithms: a case-study for scheduling *Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature (Brussels, 1992)* ed R Männer and B Manderick (Amsterdam: Elsevier) pp 229–38

——1993 Genetic state-space search for constrained optimization problems *Proc. Int. Joint Conf. on Artificial Intelligence (Chambery, 1993)* (San Mateo, CA: Morgan Kaufmann) pp 967–72

Schöneburg E and Heinzmann F 1992 PERPLEX: Produktionsplanung nach dem Vorbild der Evolution *Wirtschaftsinformatik* **34** 224–32

Smith S F 1992 Knowledge-based production management: approaches, results and prospects *Production Planning & Control* **3** 350–80

Starkweather T, McDaniel S, Mathias K, Whitley D and Whitley C 1991 A comparison of genetic sequencing operators *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, July 1991)* ed R K Belew and L B Booker (San Mateo, CA: Morgan Kaufmann) pp 69–76

Stöppler S and Bierwirth C 1992 The application of a parallel genetic algorithm to the n/m/P/Cmax flowshop problem *New Directions for OR in Manufacturing* (Berlin: Springer) pp 161–75

Syswerda G 1991 Schedule optimization using GAs *Handbook of Genetic Algorithms* ed L Davis (New York: Van Nostrand Reinhold) pp 332–49

Syswerda G and Palmucci J 1991 The application of genetic algorithms to resource scheduling *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, July 1991)* ed R K Belew and L B Booker (San Mateo, CA: Morgan Kaufmann) pp 502–8

Tamaki H and Nishikawa Y 1992 A paralleled genetic algorithm based on a neighborhood model and its application to the jobshop scheduling *Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature (Brussels, 1992)* ed R Männer and B Manderick (Amsterdam: Elsevier) pp 573–82

Uckun S, Bagchi S, Kawamura K and Miyabe Y 1993 Managing genetic search in job shop scheduling *IEEE Expert* 15–24

Whitley D, Starkweather T and Fuquay D'A 1989 Scheduling problems and traveling salesman: the genetic edge recombination operator *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, June 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 133–40

Whitley D, Starkweather T and Shaner D 1991 The traveling salesman and sequence scheduling: quality solutions using genetic edge recombination *Handbook of Genetic Algorithms* ed L Davis (New York: Van Nostrand Reinhold) pp 350–72

Yamada T and Nakano R 1992 A genetic algorithm applicable to large-scale job-shop problems *Proc. 2nd Conf. on Parallel Problem Solving from Nature (Brussels, 1992)* ed R Männer and B Manderick (Amsterdam: Elsevier) pp 281–90

Zimmermann A 1985 *Evolutionsstrategische Modelle bei einstufiger, losweiser Produktion* (Frankfurt: Lang)

Zweben M and Fox M S 1994 *Intelligent Scheduling* (San Mateo, CA: Morgan Kaufmann)

F1.6 Pattern recognition

K Govinda Char and Walter Alden Tackett

Abstract

Pattern recognition is one of the most important components of any intelligent system. The traditional methodologies in pattern recognition are inadequate to provide optimal solutions to a variety of pattern recognition and classification problems that are inherently complex. In recent years, evolutionary algorithms have been successfully applied to a wide range of diverse sets of problems in the field of pattern recognition. In a number of applications, evolutionary paradigms, in hybrid with the traditional techniques or in isolation, have outperformed traditional techniques. In this section we provide an overview of various pattern recognition techniques that are currently in use, the role of evolutionary computation in adaptive pattern recognition and the future trends.

F1.6.1 Introduction

Pattern recognition and classification are inherent components of any intelligent system. Basically pattern recognition is a general description for the family of algorithmic methods covering all aspects of information processing ranging from data perception, data acquisition, data filtering, and low-level analysis to high-level interpretation. The traditional pattern recognition methodologies include statistical, syntactic, and neural network approaches. No single approach proves consistently optimal to tackle the wide range and variety of pattern recognition and classification tasks. Evolutionary algorithms, being population-based paradigms, have been shown to successfully perform parallel search for solutions in complex problem spaces. Thus, these paradigms have yielded optimal solutions to a number of difficult problems that are intractable and in some cases almost not solvable with traditional techniques. These include problems in various domains such as *computer vision*, *image processing*, *face recognition*, speech recognition and understanding, remote sensing, medical diagnosis, and others. In this section, we introduce the basic concepts of pattern recognition through an example and discuss the various representations that are currently employed. Further, we briefly discuss the role of evolutionary computation in the context of each of these representations in automated pattern recognition and classification.

G8.1, G8.2, G8.3

We consider pattern recognition applications of the classical ‘statistical’ variety, as characterized by Duda and Hart (1973) and others. Pattern recognition takes place in a vector space where each dimension of the space corresponds to a feature of the object being recognized. The process of recognition is one in which a system chooses an appropriate label to each vector in the space, denoting the category to which it belongs. In general, this is achieved by partitioning the feature space with a discriminant function, which outputs the category as a function of feature values. Feature axes may in general consist of either discrete or continuous quantities. As a concrete example, consider a system which classifies individuals as either ‘academic researchers’ (category AR) or ‘rock stars’ (category RS) based on feature vectors comprising measurements of (i) IQ, (ii) yearly income, and (iii) hair color. These three features represent respectively a range of ordered integer values, a floating-point quantity with a large dynamic range, and a set of discrete unordered values. Each feature is subject to unique forms of objective and subjective error when it is measured for an individual sample. Therefore, an individual, regardless of category, is represented as a point in a space of three heterogeneous dimensions, possibly translated by some amount from its ‘true’ location due to measurement error.

In general, pattern recognition is achieved through observation of a set of sample values, called the training set. The basis for measuring algorithm performance is the distribution of samples correctly and incorrectly classified. In general this measure is only meaningful when applied to an independent set of samples, called the test set, due to the problem of 'overfitting' which is well known in problems of regression.

F1.6.2 Representation

Another important consideration is the representation chosen for the discriminant function. The following is a representative sample of possible representations. None of these representations excludes the use of evolutionary computation nor are any of them exclusive to evolutionary computation. Likewise, differences in classification performance between evolutionary computation and other methods are anecdotal, not theoretical. However, some representations can claim advantages and disadvantages in performance. We discuss these through the following representations.

F1.6.2.1 Clustering methods

Clustering is an important technique used in data identification and data analysis. The purpose of clustering is to partition a set of given objects into a set of clusters such that the objects in a particular cluster are more similar to each other than to objects in other clusters. A set of control points is placed in feature space, so that there is at least one point for each category. Classification of a sample point is determined by the category label(s) of the control point(s) nearest to it in some sense. The positions of the control points are adjusted so as to optimize classification performance. Basically, there are two categories of clustering algorithms: constructive and iterative algorithms. For large data sets, these algorithms are not only slow but eventually yield suboptimal solutions, thereby either sacrificing the accuracy for time complexity or vice versa. Other methods such as *simulated annealing* (Metropolis *et al* 1953) have been found to be unsuitable for clustering problems due to their excessive execution time. Evolutionary algorithms such as *genetic algorithms* (GAs) (Holland 1975, Goldberg 1989) have been applied effectively to clustering problems. As an example, using problem-specific genetic operators, Bhuyan *et al* (1991) have shown that the evolutionary approach can overcome the problems of time complexity and also the local minima, suggesting that genetic clustering methods may be very promising. Other evolutionary paradigms, such as evolutionary strategies (Rechenberg 1965, 1973), have also performed very well in cluster analysis (Phanendra Babu *et al* 1994).

D3.5
B1.2

F1.6.2.2 K-nearest-neighbor (KNN) algorithms

These algorithms are similar to the clustering methods described above, but with training set and control points the same. The class of a point in the test set is determined by polling the classes of the K training points which are 'nearest' to it according to some metric. Control-point adjustment algorithms for clustering methods are generally fast in comparison to the genetic algorithm when the data samples are small. For larger data samples and overlapping clusters, more sophisticated recognition techniques must be employed. Other classical methods such as K-means (Duda and Hart 1973) and learning vector quantization (Kohonen 1995) are essentially hill-climbing methods, and can be trapped into suboptimal solutions. Representing the control points in the GA chromosome has the potential to overcome false optima. The same can be said, however, of applying simulated annealing, which is not a method of evolutionary computation. The disadvantages with simulated annealing are the difficulty in fixing the annealing schedules and also that it is slow. In Kelly and Davis (1991), a genetic algorithm is applied to the KNN representation with favorable results, and is an excellent example of how evolutionary computation methods may be applied in hybrid with more conventional statistical methods. As another example, Punch *et al* (1993) have used the KNN algorithm within the evaluation function of a genetic algorithm for pattern classification. This approach combines feature selection and data classification and is applicable to large data sets and is also computationally less expensive compared to other traditional techniques.

F1.6.2.3 Neural networks

Artificial neural networks are biologically motivated paradigms for machine learning. In its simplest form, a neural network is a collection of activatable units that are connected through a set of real-valued

weights. Input features are treated as a vector and are subjected to a sequence of matrix multiplication, possibly with some nonlinear operations upon intermediate vectors. The output of the final operation is thresholded or otherwise processed to produce the category label. Thus, these networks are highly capable of detecting patterns and regularities of input data. With the classical approach, estimating an optimal topology for a given classification task is difficult, resulting in suboptimal solutions. Also, the inability of the learning algorithms to detect conditions of local optima is a common problem. To alleviate these problems, evolutionary methods were tried first by Montana and Davis (1989). *Evolutionary artificial neural networks* have been the focus of research in recent years (Yao 1993, Balakrishnan *et al* 1995). There exists a rich variety of algorithms for learning weights in a neural network. Many important considerations of applying genetic algorithms to weight learning are discussed by Whitley *et al* (1992, 1995) and by Belew *et al* (1991). Another aspect of evolutionary applications to neural network development is the determination of network architecture: the number and configuration of intermediate elements between input and output. Genetic algorithms have been used to search for optimal network topology (Harp *et al* 1991, Mühlenbein 1990, Polani *et al* 1993). New network learning rules have also been evolved successfully with genetic algorithms (Chalmers 1990, Dasdan *et al* 1993). However, other evolutionary paradigms such as *evolutionary programming* (EP) (Fogel *et al* 1966), cellular encoding (CE) (Gruau 1994) and *genetic programming* (GP) (Koza 1992) are found to be much more appropriate for evolving neural networks that involve structure acquisition and also the acquisition of parametric values. Recently, emergent neural networks have been evolved with evolutionary programming (Fogel 1992, 1993, Angeline 1993).

D1 B1.4 B1.5.1

F1.6.2.4 Decision trees

Decision trees are basically employed for pattern classification tasks. At the root of the tree, an individual feature is examined, and if the value falls into some subset of values, then the decision is passed to the left branch of the tree, else the decision is passed to the right branch. The tree is recursive, so that a given branch may terminate in a label, or may itself branch. Different branches use different features for their decision. Note that this describes only a naive version of the algorithm; more complex variations abound. As a simple example, ID3 (Quinlan 1986) is a hierarchical classification system for inducing a decision tree from a given set of training examples. This method of decision tree formation can often deal better with unordered discrete features than other methods, but has trouble dealing with feature types whose values are statistically correlated. ID3 can very often generalize and classify an unknown object into its correct class. Although genetic algorithms have not been applied directly for induction of decision trees, classification algorithms employing genetic algorithms to successfully induce decision trees are found in the literature (Turney 1995). The role of the genetic algorithm in this case has been to find the parameters of the classification algorithm. Recently, decision trees have been evolved successfully (Koza 1990, Gökhan Tur *et al* 1996) with an extension of the genetic algorithm, that is, genetic programming. This approach suggests improved results when compared to other decision tree induction algorithms that use greedy search methods. We discuss GP in the next section.

F1.6.2.5 Arbitrary functions

An arbitrary function, constructed from mathematical primitives and/or control structures, takes feature values as input and outputs a value which is thresholded or otherwise processed to produce the category label. This domain of representation is native to the genetic programming method of evolutionary computation. Genetic programming is basically a genetic algorithm for program discovery. Genetic programming has been shown to tackle complex, real-world problems. As an example, Tackett (1993), through empirical demonstration, has shown that the GP paradigm can be applied successfully to an extremely complex task such as *image discrimination* in automatic target recognition. Furthermore, because of its unified approach as a paradigm for evolving computer programs, any of the above representations can be induced in terms of computer programs. Recently multilayer feedforward neural networks have been evolved with genetic programming to optimize both architecture and connection weights (Zhang and Mühlenbein 1993). Gruau (1993) has developed a novel method, cellular encoding, for evolving neural networks using grammar encoding. This approach employs genetic programming as the basic evolutionary mechanism. Grammar encoding facilitates the building of compact, modular neural networks that can scale up very well with the problem size. Using building blocks (Koza 1994) in genetic programming, Char

G8.2

(1996) suggests the possibility of coevolving new learning rules for emerging structures. This approach combines cellular encoding and genetic programming paradigms. Recently, de Garis (1990) has provided a brilliant exposition of applying genetic programming for evolving artificial nervous systems and embryos, suggesting the possibility of the evolution of an artificial brain in the near future (Shimohara 1992, Vaario 1992). This development, in particular, clearly suggests a new perspective in evolving intelligent systems that can not only process information but would also generate new information. Accordingly, these developments will have a tremendous effect on present pattern recognition techniques, which happen to be just one of the components of intelligent systems. These trends indicate that highly sophisticated pattern recognition systems are bound to emerge in the future, where machine intelligence will be a dominant technology.

F1.6.3 Conclusion

We have provided an overview of the various representations employed in the field of pattern recognition and classification. The inadequacies of traditional pattern recognition techniques in tackling difficult problems and the role of evolutionary paradigms with each of these representations in improving the performance have been discussed through various examples. The advantages of using evolutionary paradigms basically stem from several facts: they are powerful search algorithms; they are easy to parallelize; and they have been shown to work on difficult problems in various domains. These developments and the recent trends suggest that evolutionary computation has great applications potential and will play a very significant role in building intelligent information processing systems in the future.

References

Angeline P J 1993 *Evolutionary Algorithm and Emergent Intelligence* Doctoral Dissertation, Ohio State University Laboratory for Artificial Intelligence Research (LAIR)

Balakrishnan K and Honavar V 1995 *Evolutionary Design of Neural Architectures—a Preliminary Taxonomy and Guide to Literature* Technical report CS TR 95-01, Department of Computer Science, Iowa State University, Ames, IA

Belew R K 1991 *Evolving Networks: Using the Genetic Algorithm with Connectionist Learning* University of California CSE Technical Report CS90-174

Bhuyan J N, Raghvan V V and Elayavalli V K 1991 Genetic algorithm for clustering with an ordered representation *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, 1991)* ed R K Belew and L B Booker (San Mateo, CA: Morgan Kaufmann) pp 408–15

Chalmers D J 1990 The evolution of learning: an experiment on genetic connectionism *Proc. 1990 Connectionist Models Summer School* (San Mateo, CA: Morgan Kaufmann)

Char K G 1996 A learning rule for emerging structures *WCNN'96 (San Diego, 1996)*

Dasdan A and Olfazar K 1993 Genetic synthesis of unsupervised learning algorithms *Proc. 2nd Turkish Symp. on Artificial Intelligence and Artificial Neural Networks (Istanbul, 1993)*

De Garis H 1990 Artificial nervous systems, artificial embryos and embryological electronics *Parallel Problem Solving from Nature (Dortmund, 1990) (Lecture Notes in Computer Science 496)* ed H-P Schwefel and R Männer (Berlin: Springer)

Duda R O and Hart P E 1973 *Pattern Classification and Scene Analysis* (New York: Wiley)

Fogel D B 1992 *Evolving Artificial Intelligence* Doctoral Dissertation, University of California at San Diego

——1993 Using evolutionary programming to create neural networks that are capable of playing Tic-Tac-Toe *Int. Conf. on Neural Networks (San Francisco, CA, 1993)* (San Diego, CA: IEEE Press) pp 875–80

Fogel D B, Owens A J and Walsh M J 1966 *Artificial Intelligence through Simulated Evolution* (New York: Wiley)

Goldberg D E 1989 *Genetic Algorithm in Search, Optimization and Machine Learning* (Reading, MA: Addison-Wesley)

Gökhan T and Halil A G 1996 Decision tree induction using genetic programming *Proc. 5th Turkish Symp. on Artificial Intelligence and Artificial Neural Networks (Istanbul, 1996)* pp 187–96

Gruau F 1993 Genetic synthesis of modular neural networks *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 318–25

Harp S A and Samad T 1991 Genetic synthesis of neural network architectures *Handbook of Genetic Algorithms* ed L Davis (New York: Van Nostrand Reinhold) pp 201–21

Holland J H 1975 *Adaptation in Natural and Artificial Systems* (Ann Arbor, MI: University of Michigan Press)

Kelly D Jr and Davis L 1991 Hybridizing the genetic algorithm and the K-nearest-neighbors *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, 1991)* ed R K Belew and L B Booker (San Mateo, CA: Morgan Kaufmann)

Kohonen T, Barna G and Crisley R 1987 Statistical pattern recognition with neural networks: benchmarking studies *Proc. Int. Conf. on Neural Networks (San Diego, CA, 1988)* (San Diego, CA: IEEE)

Koza J R 1990 Concept formation and decision tree induction using genetic programming paradigm *Parallel Problem Solving from Nature (Dortmund 1990) (Lecture Notes in Computer Science 496)* ed H P Schwefel and R Männer (Berlin: Springer) pp 124–28

——1992 *Genetic Programming: On the Programming of Computers by Means of Natural Selection* (Cambridge, MA: MIT Press)

——1994 *Genetic Programming 2: Automatic Discovery of Reusable Programs* (Cambridge, MA: MIT Press)

Metropolis N, Rosenbluth A, Rosenbluth M, Teller A and Teller E 1953 Equations of state calculations by fast computing machines *J. Chem. Phys.* **21** 1087–92

Montana D J and Davis L D 1989 Training feedforward neural networks using genetic algorithms *Proc. 11th Joint Conf. on Artificial Intelligence* (San Mateo, CA: Morgan Kaufmann) pp 762–7

Mühlenbein H 1990 Limitations of multi-layer perceptron networks—steps towards genetic neural networks *Parallel Comput.* **14** 249–60

Phanendra Babu G and Murthy N 1994 *Pattern Recognition* vol 27 (Amsterdam: Elsevier) pp 321–9

Polani D and Uthmann T 1993 Training Kohonen feature maps in different topologies: an analysis using genetic algorithms *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 326–33

Punch N F, Goodman E D, Pei min, Chia-Shun L Hovland P and Enbody R 1993 Further research on feature selection and classification using genetic algorithms *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 557–64

Quinlan J 1986 Induction of decision trees *Machine Learning* **1** 81–106

Rechenberg I 1965 *Cybernetic Solution Path of an Experimental Problem* Ministry of Aviation Royal Aircraft Establishment, Farnborough, UK

——1973 *Evolutionsstrategies Optimierung Technischer System nach Prinzipien der Biologischen Evolution* (Stuttgart: Frommann-Holzboog)

Shimohara K 1992 Evolutionary system for brain communications—towards an artificial brain *Artificial Life IV: Proc. 4th Int. Workshop on the Synthesis and Simulation of Living Systems* ed R Brooks and P Maes (Cambridge, MA: MIT Press) pp 3–7

Tackett W A 1993 Genetic programming for feature discovery and image discrimination *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 303–9

Turney P D 1995 Cost sensitive classification; empirical evaluation of hybrid genetic decision tree induction algorithm *JAIR* **2** 369–409

Vaario J 1992 *Modeling Adaptive Self-Organization* ATR Human Information Processing Research Laboratories Evolutionary Systems Department, Japan

Whitley L D and Schaffer J D (eds) 1992 *COGANN-92; Int. Workshop on Combinanations of Genetic Algorithms and Neural Networks* (IEEE Computer Society)

Whitley L D and Vose M D (eds) 1995 *Foundations of Genetic Algorithms 3* (San Mateo, CA: Morgan Kauffman)

Yao Xin 1993 A review of evolutionary artificial neural networks *Int. J. Intell. Syst.* **8** 539–67

Zhang, B-T and Mühlenbein H 1993 Genetic programming of minimal neural nets using Occam's razor *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 342–9

F1.7 The packing problem

Kate Juliff

Abstract

Packing problems encompass a broad range of optimization problems that are concerned with the placing of different-sized objects in a number of containers, subject to various hard and soft constraints. Conventional genetic algorithms (GAs) alone perform no better than traditional artificial intelligence approaches such as heuristic search or operations research methods such as First Fit. The primary limitation of traditional GA methods is their inability to accommodate the representation of solutions in such a way as to make use of the GAs' inherent power. Solutions to many packing problems do not map easily into a single-chromosome structure, and standard reproduction operators do not allow the inheritance of good parts of a solution. More promising are extensions of the original GA approach which have moved away from the single-chromosome representations to more sophisticated chromosomal models that better reflect the complexity and multidimensionality of criteria to be satisfied in finding optimal or near-optimal solutions.

F1.7.1 Introduction

One important class of combinatorial optimization problems is that of packing. Packing problems involve the placing of objects into containers in an optimal manner. Brown (1971) uses the analogy of bricks and holes as a definition of this class of problem. There exist a number of holes, and a number of bricks to pack into them. Bricks and holes may of be of different sizes. The problem is to pack all bricks with no portion protruding, as well as satisfying other constraints that are local to the specific problem. Examples of packing problems are timetabling, bin packing and stock cutting.

Packing problems can be divided broadly into three classes. In the first type of packing problem, the number of containers may be fixed and the problem is to pack as many objects as possible into the fixed number of containers. In the second type, the number of objects is fixed and the problem is to pack all objects into a minimum number of containers. These bin packing problems have the single hard constraint that the sum of sizes of all objects in any one bin cannot exceed its capacity.

In the third class of packing problem, the number of objects and the number of containers may both be fixed, and the problem is to optimally pack all objects into the containers such that a number of hard and soft constraints are satisfied. Here the problem is not to pack as many objects as possible, but to optimally arrange the objects into a given number of containers. Timetabling and three-dimensional freight loading problems are examples of this type of packing. In school timetabling for example, hard constraints include that no class may be held outside certain hours, that no teacher or pupil can be assigned more than one class at any one time, and that all classes are held. Soft constraints may be that workload is evenly distributed and that certain classes are held in certain rooms. This class of packing problem is similar to *scheduling problems* where events are packed in time. Because of the similarity between the two types of problems, results of research in scheduling are often relevant to packing research. F1.5, G9.4

Evolutionary computation methods have been applied to packing problems in the form of a number of types of *genetic algorithms* (GAs). Early attempts to apply GAs have not been overly successful, due to problems in the representation of solutions into artificial genetic structures, and to the limitations of the traditional genetic operators. More promising are those techniques that apply nontraditional genetic approaches or that combine GA techniques with other methods. B1.2

F1.7.2 Evolutionary computation approaches

F1.7.2.1 Traditional genetic algorithms

A traditional GA, for any problem, uses a single chromosome to represent a solution. Each gene in the chromosome represents an object to be packed and the containers are not explicitly represented. Objects are assigned to containers by their position on the chromosome which is an abstraction of a queue of objects in line to be packed. This approach has variations. There may be a number of chromosomes in an individual solution, all of the same structural type, each representing groups of objects, with the solution being a set of chromosomes, one per group or container. But in all cases, a single gene represents one object and all chromosomes in the population of solutions are identical in structure. Performance of this type of GA has not been spectacular and such types have generally been tested with very simple packing problems. The poor performance is largely due to the inappropriateness of the single-chromosome GA and its operators for representing packing solutions. It is worth examining some of these GAs as they illustrate those problems inherent in the traditional approach.

Direct mapping genetic algorithms. Classic GAs use an encoding where the chromosome is a literal or direct representation of the solution. Packing problems may be represented directly by mapping items to be packed to individual genes, such as in the *traveling salesman problem* where cities are represented as genes and the solution is decoded as the order of genes on the chromosome. G9.5

An example is the directly mapped GA designed by Abramson (1991) to solve school timetabling problems. A timetable is represented as a number of tuples, each of which contains a class number, a teacher number and a room number identified by a label. A period in a day is represented as a list of labels. Simple *one-point crossover* leads to invalid solutions where duplicate classes are represented and some classes are not represented at all. To overcome this problem, Abramson used a specialized mutation operator so that missing classes would be replaced and duplicate classes would be removed. Using a limited number of timetabling constraints, his GA found near-optimal and optimal solutions on test data for complex problems although reported execution times were quite slow. C3.3.1

There are two drawbacks to this approach. Firstly, the search space is larger than it has to be, and thus invalid solutions are found and evaluated. Even though invalid solutions are corrected by a specialized mutation operator, the power of genetic search is reduced as time is spent searching for and eliminating solutions which cannot be considered. Secondly, the need for specialized operators means that new operators have to be devised for each packing problem.

Not all features to be optimized are represented in the search. In Abramson's timetabling GA each class has a number of attributes (class, teacher and room). However, these attributes are not represented explicitly in the chromosome. There is no search, therefore, of best combination of teacher, class and room, as there is no way to represent these attributes in the single-chromosome string.

Indirect mapping genetic algorithms. In order to avoid the use of specialized operators in packing and scheduling GAs, a number of researchers (Davis 1985, Prosser 1988, Syswerda 1991) have used indirect mapping. With such mappings the chromosome specifies parameters to be interpreted by a decoder which takes information from the chromosome and builds a valid solution according to parameters encoded in the chromosome.

In packing GAs, the chromosome represents a queue of objects to be packed. Standard order-based reproduction operators, which are not problem specific, can be applied to ensure that all objects are represented in the queues once and once only.

Prosser used indirect mapping and a decoder for a pallet packing problem. His problem involves stacking pallets with a set number of different-sized metal plates such that a minimum number of pallets is used. Performance compared favorably with that of a branch-and-bound algorithm, in terms of quality of solutions and execution time. Problem sizes were small, with items to be packed varying from 10 to 40 items.

Although indirect representation and order-based crossover overcome the problem of invalid solutions, there remain the problems of redundancy of solution and the failure to map all aspects of a problem in a single chromosome. A number of approaches have been developed to address these problems. Some of these approaches concern scheduling GAs which suffer from many of the difficulties of packing GAs.

F1.7.2.2 Nontraditional genetic algorithms

Schemes where the chromosome represents a queue work well when the ordering or grouping problem is one-dimensional in the sense of features to be mapped. In Prosser's packing problem (Prosser 1988) for example, pallet order alone needs to be represented. However, when there is more than one feature to be optimized this approach is limiting, as only the feature represented in the chromosome is optimized.

In timetabling problems, information about teacher, pupils, rooms and times needs to be represented. If only one feature of the solution is represented by the chromosomes, then other methods outside the GA must be used to search those areas of the search space not sampled by the GA.

Uckun *et al* (1991) describe this drawback as a problem of constrained search space. They tackled this problem in a job-shop scheduling GA, by representing additional information in a single chromosome. Although this research was applied to job-shop scheduling, it is of interest here because it was an attempt to overcome the limitations of a single-chromosome GA.

Bagchi *et al* (1991) developed three GAs with indirect mapping to tackle a simple job-shop problem. One GA represents job order alone and the rest of the search space is searched by a schedule builder. This GA was compared with two other GAs where other features of a solution (job order, process plan and machine) were represented by a single chromosome. What is of interest in their work is that they recognized the problem of 'constrained search space' and attempted to overcome it. However, because a single chromosome represented a multifaceted solution, specialized reproductive operators were required. On hypothetical job-shop problems the GA searching all three features produced the most successful results at the expense of longer execution times.

Kröger *et al* (1991) also recognized the importance of encoding in their work on packing two-dimensional rectangles. In order to represent both location and orientation of rectangles in a limited space, they used a binary tree with each node representing one rectangle. Specialized genetic operators were developed to ensure inheritance of good features of solutions. Results were promising for problems involving up to 30 rectangles.

Grouping genetic algorithms. In an attempt to overcome the problem of redundant search for a class of packing that he describes as 'grouping problems', Falkenauer (1991) developed the Grouping Genetic Algorithm (GGA). Falkenauer (1991, 1992, 1994) applies GAs to a number of problems including bin packing. The bin packing problem is represented as a problem of grouping items into categories, with categories corresponding to bins. Chromosomes represent categories, rather than the objects that they contain. The items to be packed form a queue and the chromosome represents the bins into which the items in the queue will be packed. Falkenauer (1994) compares the grouping GA approach with a heuristic (First Fit Descending). Results show the GGA to be superior in performance to the heuristic, especially in more difficult cases. However, even more superior results were found when the GGA was used in conjunction with local optimization.

Falkenauer's method works well with bin packing problems. The order of objects within bins and the order of the bins are of no importance. A more complex problem is a packing problem where objects must be assigned to containers, and, within each container, objects must be assigned a position. Many packing problems involve several hard and soft constraints, and thus a successful solution involves not one criterion but several.

Multichromosome genetic algorithms. In nature only very simple organisms are represented by single chromosomes and more complex organisms have many. By representing a packing solution as a queue of objects or a set of containers, only one feature is represented. Others must be found by the intelligent decoder. If the full power of the genetic search is to be exploited, it is necessary that all features of the solution are mapped to the chromosomal structure.

The multichromosome GA was developed (Juliff 1993) to represent a number of features of a solution (pallet order, carton grouping and pallet type) in a complex pallet packing problem. The problem was mapped to three chromosomes, each representing one of these features. Each individual solution was represented by three independent chromosomes. Reproduction was implemented as in nature, by taking one chromosome from each parent to make three chromosome pairs and applying crossover and mutation separately to each pair.

Figure F1.7.1 shows a generic design for a chromosomal representation of a packed truck, similar to Juliff's multichromosome GA. In this example the queue representation (representation A) is compared with

a representation (B) where three chromosomes represent three different features of a solution, blueprinting a single individual. Chromosomes 1 and 2 represent carton order and pallet order, respectively. They are queues of cartons to be loaded onto pallets, and pallets of these cartons to be loaded on to a truck. The third chromosome represents the grouping of like cartons together, with the number held by each gene representing the number of cartons in each carton type. The first two chromosomes are order-based, whilst the third is a specialized chromosome representing a feature specific to the loading problem. Note that the single chromosome in representation A, and chromosomes 1 and 2 in representation B, fulfill the same function. Both act as a queue to a decoder that packs items according to their position in the queue. However, that is all that can be done by the single-chromosome GA. The multichromosome GA, on the other hand, encodes other information (pallet order and pallet type) that is thus optimized by the GA, and has been tested on problems using real data obtained from an Australian manufacturer. Load sizes varied from 32 to 112 units, and the results were compared with the same problem using a single-chromosome GA with queue-based representation. The multichromosome GA clearly outperformed the single-chromosome GA version even when the latter was given an initial population of semi-optimized individuals. By using more than one chromosome per individual solution, more features of an individual can be represented. Just as in nature, where the evolution of more complex organisms requires more complex genetic encoding, in the GA approach, complex solutions require more than a single-dimensional chromosomal string.

F1.7.3 Non-genetic-algorithm approaches

The GA approach to optimization problems has generally arisen from the failure of traditional methods to overcome the problem of local minima. Traditional approaches to packing problems, such as iterative and branch-and-bound searches, face this problem. A configuration may be found that is better than its near neighbors, but is not the global best. In attempting to overcome this problem three stochastic search methods have emerged, one of which is the GA method. The other two stochastic methods, the Hopfield and Tank (1985) *neural network* and simulated annealing, have a number of commonalties. D1
All of these methods require a global evaluation function. That is, completed solutions are 'found' and evaluated. In contrast, traditional methods of iterative search and branch-and-bound evaluate and build on partial solutions. All three methods involve sampling of the entire solution space, in an attempt to avoid entrapment in a local minimum. The three methods consider poor solutions as well as good ones on the basis that a poor solution may yet have qualities that will form part of another, better solution.

Although the author has been unable to find any comparative studies of the three stochastic search methods specifically applied to packing problems, a number of researchers have compared the methods using other types of optimization problem (Peterson 1990, Spears 1990, Ulder *et al* 1990). Their research indicates that GAs are promising as a stochastic search method for optimization problems, especially when combined with other methods, such as local search.

F1.7.4 Conclusion

GAs are a promising method of solving packing problems and they compare favorably with other stochastic search methods which have been developed to overcome the local minima problems inherent in AI and operations research approaches. However, traditional GAs need to be combined with other methods in order to achieve good results.

The main problem is their inability to map complex solutions in a single string. This consideration of mapping the problem to the GA's representation of the search space is especially important when the problem to be solved is multidimensional in the sense that it involves the representation of a number of different features of a problem. Recent work indicates that if the traditional one-chromosome structure is replaced by more appropriate mapping, such as in a grouping or multichromosome GA, a number of the problems associated with poor performance can be eliminated. More work needs to be done and care taken in choosing the right chromosomal representation in order that packing problems, especially complex ones, can be successfully tackled by a GA approach.

References

Abramson D 1991 *A Parallel Genetic Algorithm for Solving the School Timetabling Problem* Information Technology Technical Report TR-DB-91-02, Royal Melbourne Institute of Technology

Bagchi S, Miyabe Y and Kawamura K 1991 Exploring problem specific recombination operators for job shop scheduling *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, July 1991)* ed R Belew and L Booker (San Mateo, CA: Morgan Kaufmann) pp 13–7

Brown A R 1971 *Optimum Packing and Depletion: The computer in space-and-resource-usage problems* (London: Macdonald)

Davis L 1985 Applying adaptive algorithms to epistatic domains *Proc. Int. Joint Conf. on Artificial Intelligence* ed A Joshi and R Gutiirrez (San Mateo, CA: Morgan Kaufmann) pp 161–4

Falkenauer E 1991 A genetic algorithm for grouping *Proc. 5th Int. Symp. on Applied Stochastic Models and Data Analysis* ed M J Valderrama (Singapore: World Scientific) pp 199–206

——1992 A genetic algorithm for bin packing and line balancing *Proc. 1992 IEEE Int. Conf. on Robotics and Automation* (Los Alamos, CA: IEEE Computer Society Press) pp 1186–93

——1994 A new representation and operators for GAs applied to grouping problems *Evolut. Comput.* **2** pp 123–44

Hopfield J J and Tank D W 1985 Neural computations of decisions in optimization problems *Biol. Cybern.* **55** 141–2

Juliff K 1993 A multi-chromosome genetic algorithm for pallet loading *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 467–74

Kröger B, Schwenderling P and Vomberger O 1991 Parallel genetic packing of rectangles *Proc. 1st Workshop on Parallel Problem Solving from Nature (Dortmund, 1990)* ed H P Schwefel and Männer (Berlin: Springer) pp 160–4

Peterson C 1990 Parallel distributed approaches to combinatorial optimisation: benchmark studies on travelling salesman problem *Neural Comput.* **2** 261–9

Prosser P 1988 A hybrid genetic algorithm for pallet loading *Proc. 8th Int. Conf. on Artificial Intelligence* ed Y Kodratoff (London: Pitman) pp 159–64

Radcliffe N J 1991 Forma analysis and random respectful recombination *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, July 1991)* ed R K Belew and L B Booker (San Mateo, CA: Morgan Kauffman) pp 222–9

Spears W 1990 Using neural networks and genetic algorithms as heuristics for N-P complete problems *Masters thesis* George Mason University, Fairfax, VA

Syswerda G 1991 Schedule optimization using genetic algorithm *Handbook of Genetic Algorithms* ed L Davis (New York: Van Nostrand Reinhold)

Uckun S, Bagchi S, Miyabe Y and Kawamura K 1991 Managing genetic search in job shop scheduling *IEEE Expert* October 1991

Ulder N L J, Aarts H L, Bandelt H, van Laarhoven P J M and Pesch E 1990 Genetic local search algorithms for the travelling salesman problem *Proc. 1st Workshop on Parallel Problem Solving from Nature (Dortmund, 1990)* ed H-P Schwefel and R Männer (Berlin: Springer) pp 109–16

Underbrink A J Jnr and Williams G Jnr 1994 Genetic algorithms applied to assembly operations *Proc. 1994 SPIE Conf. on AI Systems in Aerospace and Industry* (Huntsville, AL: Boeing) pp 1–11

F1.8 Simulation models

Ulrich Hammel

Abstract

In this section we briefly discuss the application of evolutionary computation in the context of simulation studies. We enumerate a set of characteristic problems that are frequently faced in the optimization of simulation models. None of these problems, however, is limited to this application domain. The first two subsections are intended to give a rough idea about the subject of simulation and to motivate the use of evolutionary algorithms. The next subsection discusses some of the critical questions in more detail. These include the genetic representation, measures to accelerate the search process, and the problem of stochastic fitness functions. We purposely omit a separate discussion of specialized genetic search operators in order to avoid a space-consuming in-depth discussion of concrete applications. This subject is treated in Part G. On the other hand, the choice of search operators is more or less determined by the design of the genetic representation which is addressed here. Finally, we summarize some alternative approaches. We argue that the advantage of evolutionary algorithms over most alternative approaches derives from their adaptability, inherent parallelism, and robustness. In order to avoid the overhead of introducing formal definitions the presentation is kept rather informal.

F1.8.1 Simulation

The intelligent design and management of complex natural and artificial systems plays a fundamental role in the lasting stability of our ecological and economical environments and therefore in the stability of our societies. Typical examples are the management of food resources such as fishing policies, the design and management of technical installations such as power plants and chemical factories, and the management of national and international economics, to mention only a few.

A prerequisite for the treatment of a complex system is the analysis of its structure and dynamics. During the last decades systems analysis has evolved into a scientific discipline in its own right, its most important tools being data analysis, simulation, and optimization.

A first step in the analysis of a given system is to identify the important observables and to gather a sufficiently large set of observations. If no additional knowledge (e.g. physical laws driving the system) is available the next step is to build up a descriptive model by mathematical or statistical means. (The discussion of physical, e.g. mechanical, models is beyond the scope of this section.)

The description of celestial mechanics by the German astronomer, Kepler (1571–1630), based on the observations of the Dane, Tycho Brahe (1546–1601), is a good example of a descriptive model: it rather precisely describes the movement of the planets but does not explain the underlying mechanics.

Such models usually possess a set of free parameters which have to be estimated by fitting the model to a set of observed data. This process (i.e. the minimization of the deviation between model generated and observed data) is called system identification. Thus, system identification can be classified as an optimization problem, which, depending on the model structure and the deviation criteria, may be hard to solve. Methods of evolutionary computation can be successfully applied here. A detailed discussion of identification is left to Section F1.4. F1.4

A descriptive model may serve to predict the behavior of a system with a certain precision, but it does not have the expressive power to assess the implication of different boundary conditions (e.g. 'What if the mass of the sun were to decrease?'). To answer this question we need the concept of gravity, that is, the interaction of the system components, a Newtonian model.

If we can make some reasonable assumptions about these interactions (e.g. physical laws such as the law of gravity), we may start to build up a *simulation model* based on these assumptions. The art of modeling consists in formulating a model which resembles the system under investigation as closely as necessary to obtain the required results but is still simple enough to be handled.

Simulation models are used as substitutes for real-world experiments for several reasons. Experiments are often too expensive (e.g. crash tests on vehicles) or too dangerous (e.g. critical states of an aircraft) to be carried out on the physical system itself. In other cases, we might even not be able to control the critical parameters at all (e.g. climate dynamics) or the time scale might not be acceptable (e.g. ecosystems).

Since powerful computer systems are available, computer-based modeling becomes more and more popular, which in turn requires formal model specifications. One common formal language is that of mathematics.

Consider the following two simple examples of *dynamical* models, both of which are taken from the book by Law and Kelton (1991) but can be found in many introductory books on simulation and related fields.

The first one represents a very idealized model of the interaction of two biological species, a predator population x of size $x(t)$ and a population y of size $y(t)$ of prey at a given time t. Let r be the reproduction rate of the prey and s the decay rate of the predator, when both are kept in isolation of each other. The probability of a 'predator catches prey' event is considered to be proportional to the population sizes. Thus, the dynamic evolution of the two populations could loosely be described in terms of two differential equations:

$$\begin{aligned} \frac{\mathrm{d}x}{\mathrm{d}t} &= -sx(t) + ax(t)y(t) \\ \frac{\mathrm{d}y}{\mathrm{d}t} &= ry(t) - bx(t)y(t). \end{aligned} \tag{F1.8.1}$$

The second model is about ordering strategies in an inventory system. For simplicity consider a company that sells one single commodity for which it is running an inventory. The time lags between single customer demands and the amounts of these demands are assumed to be independent random events, as well as the lag between placing an order to the company's supplier and the time of arrival. The monthly inventory costs are composed of the holding costs $I^+(t)$ and the shortage losses $I^-(t)$ due to the dissatisfaction of customers in case of delivery problems. The overall average monthly costs over a period of time n is given by

$$I^+ + I^- = \frac{\int_0^n I^+(t)\,\mathrm{d}t}{n} + \frac{\int_0^n I^-(t)\,\mathrm{d}t}{n} \tag{F1.8.2}$$

and obviously depends on the sequence of actual events and the policy run by the company to place orders of amount Z. This policy is assumed to be

$$Z = \begin{cases} S - I & \text{if} \quad I < s \\ 0 & \text{if} \quad I \geq s \end{cases} \tag{F1.8.3}$$

where I is the current inventory level and s, S are called decision variables.

Once a model is formulated, we should in principle be able to calculate the evolution of the model's state over time given an initial state $S(t_0) = (x(t_0), y(t_0))$ or $S(t_0) = I(t_0)$ and the set of parameters (a, b, r, s) or (s, S), respectively. This process is called *simulation*. Simulation usually requires special techniques depending on the type of model. For instance, in the case of differential equations, numerical integration techniques such as Runge–Kutta methods are applied, whereas for *discrete event models*, such as the inventory model, a runtime system is needed which supplies random number generators, queueing of events, and the like. Note that a single simulation run of a stochastic model, as in the case of the second example, is not necessarily significant for the average behavior of the model. We will return to this problem in section F1.8.3.3.

Simulation studies are used to explore consequences of measures and variations on the model (e.g. 'What is the effect of changing the initial state S_0 or changing some of the critical parameters?'). However,

in most cases the implicit goal is to find an instance of the model exhibiting a behavior that we evaluate as optimal according to some predefined criteria, such that the original system can be tuned according to the simulation results (e.g. 'What are the parameter settings for a cost-minimal order policy?'). Now our focus is on the search for an optimal setting of decision variables and this is where evolutionary computation comes into play.

In order to define an optimization problem $f(\boldsymbol{x}) \rightarrow \min$, where f is based on a simulation model, such that $\boldsymbol{x}$ is a vector of model parameters, care has to be taken in the design of the objective function f. In the case of dynamical simulation models, it is often appropriate to define $f(\boldsymbol{x})$ as an integral over a given period of time similar to equation (F1.8.2), but often it is necessary to formalize our goals more precisely, for example, to cut off the initial transients or to weight different observables. If multiple conflicting objectives have to be accounted for, *vectorial optimization techniques* (Fonseca and Fleming 1995) can be applied. C4.5.3

As introductory textbooks to the field of modeling and simulation, we recommend those by Law and Kelton (1991) and Kheir (1988). For a system-oriented approach see, for example, the book by Klir (1991).

F1.8.2 Simulation and optimization

It should be obvious from the previous subsection, that complex simulation studies may lead to nontrivial optimization tasks, but what is special about the optimization of functions based on simulation models? Experience from diverse applications tells us that we are usually faced with a subset of typical problems which justifies a separate discussion of this very subject. On the other hand, none of these problems is limited to the field of simulation. In this sense the overlap with other fields, such as *optimal control*, *identification*, and experimental *optimization*, is considerable. F1.3 F1.4, F1.2

In this section, we extract the key points of optimization in case of simulation studies. Pointers to solutions are given in the next section.

First of all, real-world simulation models are generally much more complex than the examples presented in the previous subsection. Many special purpose simulators have been developed, where only few of them have been equipped with optimization software. Thus, if optimization techniques shall be applied to the repeated design of experiments, we have to build some kind of optimizer around the simulator at hand. Since only in rare cases is the simulator's internal representation of the model accessible, for example, in order to impose self-designed search algorithms or to gain analytical information such as derivatives, the optimization strategy can only be based on the pure input–output behavior of the simulation model, which in this sense is viewed as a black box.

Second, in most cases a single simulation run is expensive with respect to resources such as CPU time, data storage, and special purpose hardware and software (including, e.g. the simulator licenses).

Furthermore, typical response surfaces (topology of the parameter space) of the associated fitness functions have properties where many traditional methods fail (see section F1.8.4). These properties include high-dimensional search spaces, multimodality, discontinuities, narrow valleys, and plateaus, to name a few.

Besides this, practical problems occur through the mixture of parameters of different types. We can easily imagine combinations of real-valued and discrete parameters from the previous examples. However, even the structure of the model itself might be subject to optimization.

In order to exemplify this problem we might have a look at a more concrete simulation project, which the author was involved in for some time. This project dealt with the optimization of chemical engineering processes. Consider a plant consisting of several process units such as reactors or distillation columns connected by streams of fluids with different temperatures and energy loads. The chemical processes require special supply temperatures, such that the fluid streams have to be preheated or cooled down before entering a process unit. Thus, a single fluid stream has to be heated up and cooled down several times. Because energy input is expensive, the goal is to exchange energy among these streams in an optimal manner, minimizing the need for external heat sources and sinks. The energy transfer from a hot stream to a cold stream is achieved by special heat exchanger devices. Since these devices introduce extra investment costs an economic optimum is searched for rather than an energetic optimum.

As a simple example, consider the heat exchanger network depicted in figure F1.8.1, where the process units are left out for simplicity. The problem consists of two hot streams h_1 and h_2 and two cold streams c_1 and c_2. The source temperatures, energy loads, and destination temperatures are given. Now different

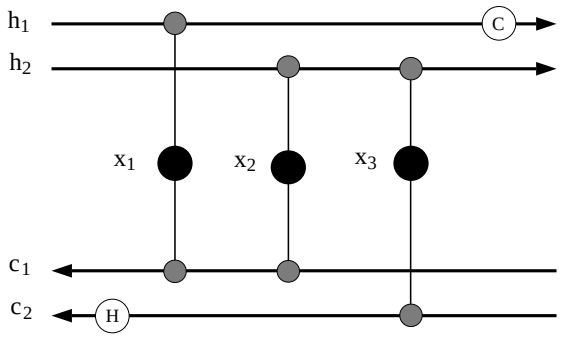


Figure F1.8.1. A sketch of a heat exchanger network with two hot streams h_1 and h_2, two cold streams c_1 and c_2 and three heat exchangers x_1, x_2 and x_3. H and C denote additional heaters and coolers.

models have to be tested (simulated) in order to identify a cost-optimal arrangement. The figure shows one arbitrary instance of such a model, introducing three heat exchangers x_1, x_2 and x_3. The point is that the goal is not limited to finding a cost-optimal setting for the heat exchanger parameters (heat loads), but the structure of the model is subject to variations, too, in the sense that heat exchangers may be added or removed and the streams might be split at some points.

Developing a representation of such a problem on which an optimizer may operate with good results is not straightforward and will be discussed in the next section, but it should be emphasized that with some additional thought in most cases it is possible to adapt the general evolutionary search heuristics to such problems and it is one of the merits of these algorithms that they are composed of rather simple and adaptable building blocks that nevertheless perform well in many environments.

Finally, a crucial point is that most of the simulation problems are stochastic in nature like the inventory model of the last section. We find another good example if we consider evolutionary algorithms as simulation models of natural processes. The application of evolutionary techniques to the search for optimal parameter settings for evolutionary algorithms is called *metaevolution*. C7.2

A single observation of a stochastic process has little significance. As a consequence, it makes no sense to base the definition of the merit function solely on a single simulation run. (In fact this is not always true since *perturbation methods* have been developed in order to derive gradient information from a single run, but this is still a field of active research, and the general applicability is questioned. Please consult the literature on simulation.) We might use the average of a statistically relevant set of observations instead. Nevertheless, the merit function still remains stochastic. Thus, we observe different fitness values for identical parameter settings. The optimization strategy has to be robust in this sense, and, according to our experience, most of the variants of evolutionary algorithms fulfil this requirement (see section F1.8.3.3).

F1.8.3 Applying evolutionary computation

There are several reasons to consider evolutionary algorithms as candidates for simulation model optimization. First of all, the absence of analytical information and the complexity of the *fitness landscapes* B2.7 (and constraints as well) demand direct search strategies with global search characteristics. However, what distinguishes evolutionary algorithms from most alternative strategies is their adjustability to a more path-oriented or volume-oriented search by changing only a few parameters, such as the *population size* E1.1 or *selection scheme*. This allows for the scaling of the algorithms according to the characteristics of the C2 problem and the availability of resources on the other hand. Furthermore, evolutionary search heuristics are so general that they can be adopted to most search spaces rather easily. Their inherent parallelism allows for reasonable response times even in the case of time-consuming simulation runs, and finally, but extremely importantly, they are robust in the case of stochastic merit functions.

In what follows, we briefly address some of the problems mentioned so far. The first problem, which we call the representation problem, deals with the question of which parts of the model should be represented on the genotype level and how this should be done in order to define appropriate genetic operators. The next section addresses the problem of resource-consuming evaluations of the merit function.

Most of the simulation problems are stochastic in nature like the inventory example of section F1.8.1. This topic will be discussed in the third section.

Only a few publications can be found in the literature dealing explicitly with evolutionary computation techniques in the context of modeling and simulation. For example, Hahn *et al* (1992) report on the calibration of a Monte Carlo simulation model by means of genetic algorithms with special emphasis on the stochastic nature of the resulting optimization problem. (See the book by Alander (1994) for a collection of references as well as that by Schwefel (1995).)

F1.8.3.1 The representation problem

In some cases, the choice of the genetic representation is straightforward, as in the case of the predator–prey model or the inventory model of section F1.8.1. In the first case, we would probably choose a *real-valued vector*, whereas the integer parameters of the second model suggest an *integer* or *binary representation*. C1.3, C1.2 Of course, all these representations are equivalent in the sense that we can define one-to-one mappings between them. (On a digital computer every object is compiled down to a digital representation anyway.) Thus, in principle we can define genetic operators on bitstrings which behave exactly such as their integer counterparts. However, if we consider a bitstring representation, we implicitly think of operators which have no knowledge about the coding of phenotypic objects: the information of whether a gene represents a real value or an integer value is lost.

In most real-world applications, such as the chemical engineering model of section F1.8.2, the choice is not that obvious and according to our experience there is no general deductive pathway for this decision, but there are a few rules of thumb.

First of all, we have to ensure that the whole parameter space will be covered by the representation: every valid structure in the example above must be representable. This sounds self-evident, but for complex parameter spaces it might be worthwhile to give some thought to this topic. For example, in the chemical engineering example above, it might be allowed to introduce splittings of the streams, such that heat exchangers are connected in parallel as sketched in figure F1.8.2.

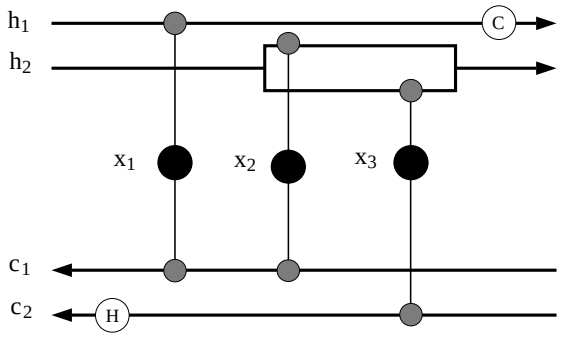


Figure F1.8.2. A sketch of a heat exchanger network including splits.

Second, try to restrict the genotype space such that invalid phenotypic objects are avoided; for example, do not allow for connections between two separate hot streams. More subtle, try to avoid the representation of models contradicting the laws of thermodynamics such as energy flows from a cold stream to a hot one. This is not always possible; for example, we might be able to detect some violations only by running the simulator.

This leads us to the subject of *constraint handling*, which is discussed in Chapter C5 in more detail. C5 Thus, we restrict this discussion to some general remarks. In the case where we are able to define some kind of metric on the infeasible region, that is, a distance measure to the feasible region, then penalty functions might be appropriate, but often the simulation run will just be terminated by the simulator issuing an error message that does not tell us anything about the grade of constraint violation. In such cases, disposing of 'lethal' individuals might be the only useful strategy. Since simulation runs are expensive, we should try to avoid wasteful simulations. If sufficient knowledge about the problem is available, it might be possible to design constraint preserving operators or repair algorithms. This has to be done with care in order not to bias the search process.

In what follows, we enumerate some general considerations for the design of genetic representations explained by the heat exchanger example.

First of all, we can apply a homogeneous basic representation ($\mathbb{B}^m$ or $\mathbb{N}^{m'}$ in our example) where all parameters (energy load and positions of connections) are coded as integers or binary strings. This allows for employing standard problem-independent *search operators*. The problem here is the construction of the coding function possibly introducing additional nonlinearities and thus hindering the search process. Furthermore, for the construction of constraint preserving operators or repair mechanisms much has to be known about the parameters' context and the coding function. For example, if a violation of a thermodynamical constraint is detected, it has to be established which of the heat exchangers are involved, the location of their corresponding 'genes', their coding function, and so on. Thus, repairing might turn out to be a complex task. C3

A step towards a more problem-related representation would be a composition of different standard representations, e.g. $\mathbb{R}^n \times \mathbb{N}^{4n}$, where each of the n heat exchangers is represented by a heat load value and four integers determining the associated streams and points of connection. Such a *mixed-integer representation* allows for the application of more specialized search operators than a pure binary representation, which in turn usually results in a better performance of the overall search process. C1.8

In some cases, that is, when the problem is separable, a decomposition of the search spaces may help. For example, structure evolution and parameter optimization are separated and eventually performed by completely different strategies. This approach is frequently proposed for structure optimization (Lohmann 1992, Rechenberg 1994) and provides good results especially if built-in strategies of the simulator can be utilized. The approach resembles a coordinate search where steps along different coordinate axes (inside separated search spaces) are iterated. However, in cases where the problem is not separable, this may lead to a poor performance of the algorithm and even to convergence to local optima.

Consider a recombination operator defined on the basis of a mixed-integer representation which blindly mixes up two sequences consisting of real and integer values. In our example this will rarely result in a feasible structure. This is due to the fact that the sequence of genes does not reflect the connectivity inside the model. To overcome this problem, we have to equip our operators with additional knowledge. This can best be achieved by developing a representation which conserves the model structure, that is, which defines a data structure reflecting the graph of figure F1.8.2. The idea is now to define recursive genetic operators knowing about the relationships between heat exchangers and streams.

Though there still remains much to say, especially about implementation techniques, we end this discussion by remarking that also *variable-length representations* can be successfully applied here. Just imagine that the algorithm should also detect the optimal number of heat exchangers rather than search for an optimal structure given a fixed set of heat exchangers. C3.4.2

The representation problem is addressed in numerous publications (e.g. Michalewicz 1994, Davis 1991).

F1.8.3.2 The resource problem

The limiting factors for the choice of strategy variants are the resource utilization of a single simulation run and the availability of resources on the other hand. In restrictive situations, simple path-oriented strategies, such as the original $(1 + 1)$ strategy of Rechenberg and Schwefel (see *evolution strategies*) might be applied first. If population-based methods are preferred, recall that the 'canonical' genetic algorithm usually requires a much larger population size than those variants based on evolution strategies or *evolutionary programming*. On the other hand, self-adaptation of strategy variables might not perform well if the *population size* is too small. An exogenous schedule of strategy parameters might then be appropriate (see *mutation parameters*). The *selection pressure* is highly responsible for the scaling of the algorithm according to convergence velocity and convergence reliability. If the number of possible runs is small, choose a selection scheme with high selection pressure. This usually leads to suboptimal solutions which is the best you can expect in the case of very limited resources. B1.3, B1.4, E1.1, E1.2, C2.1

In some situations we can apply approximation techniques to circumvent, or at least reduce, the 'evaluation bottleneck'. One approach is to replace the simulation model partially by a substitute function which resembles the response surface of the original model evaluation. In the chemical engineering example above, we designed a substitute function on the basis of the overall energy consumption. Furthermore, some simulators, especially those built on numerical approximation techniques, allow for the control of the approximation precision. Reducing the precision level usually results in a considerable

speedup of the single simulation run.

Both techniques can be utilized to guide the search during the exploration phase of the process. This is done by initially using the substitute function and/or a low precision level for fitness evaluation. During the search the substitute function will then be more and more replaced by the original simulation model and/or the precision level is increased constantly. The error introduced by the approximation can be viewed as a perturbation of the fitness function which will be discussed in the following section. The first technique for instance is mentioned by Fitzpatrick and Grefenstette (1988).

Most promising to gain speed is the exploitation of the inherent parallelism of evolutionary algorithms. This subject again could easily fill a whole chapter and is addressed by several sections distributed over this handbook.

Considerable speedup can be gained from parallelizing the search process: the single simulations must run in parallel. In order to do so some technical requirements have to be met. Usually, general purpose simulators are not available for special parallel hardware architectures. Then, we are restricted to networks of workstations and the like. But be aware that parallel execution of large simulations using sophisticated simulators might burden the overall system by a huge amount of memory and disk space consumption as well as I/O operations. Furthermore, the workloads on single nodes have to be scheduled. Next, a decision has to be made about the communication software on which the parallel algorithm can be based. In our experience PVM (parallel virtual machine) (Geist and Sunderam 1992, Bäck *et al* 1995) is a good choice.

Due to the resource demand we are almost always restricted to coarse-grained parallelism. For a simple architecture design we propose a master–slave approach, where the slaves are responsible for fitness evaluations, i.e. running the simulator, whereas the evolutionary algorithm resides inside the master process, but more elaborate *population structures* can also be built upon this architecture. C6

Finally, we have to think about the selection scheme to be applied. In the case where we are running a small population, i.e. the size does not considerably exceed the number of available machines, we would probably watch a number of nodes running idle if we impose a fixed population size and a selection scheme based on the complete set of fitness values of the whole population. This is especially true if single simulation runtimes vary dramatically, which is often the case due to premature termination of runs caused by constraint violations, usage of substitution functions, varying machine workloads, and the like. Variable population sizes, steady-state selection, or *tournament selection* may circumvent this problem. C2.3 For further reading follow the links and consult the referred literature.

F1.8.3.3 Stochastic optimization

As already mentioned, in the case of stochastic models the merit function has to be based on a set of observations rather than on one single experiment. (We omit the discussion of stochastic constraints which is a topic in its own right.) In general, we can define the resulting *stochastic optimization problem* as

$$F(\boldsymbol{x}) \rightarrow \min \tag{F1.8.4}$$

where

$$F(\boldsymbol{x}) = E[\![f(\boldsymbol{x}, \omega)]\!] = \int f(\boldsymbol{x}, \omega) P(\mathrm{d}\omega)$$

and ω is an element of some kind of probability space. Of course, other measures of the underlying distribution can be incorporated into the objective function as well. The expectation $E[\![\,]\!]$ has to be approximated by computing the mean of a sample of observations of size t.

For further reading on stochastic optimization, we recommend the books by Ermoliev and Wets (1988) and Kall and Wallace (1994).

Besides some application-oriented investigations (Hahn *et al* 1992, Hampel 1981), little work has been done on the application of evolutionary computation to general stochastic optimization problems, but some results (Rechenberg 1994) are known for noisy objective functions, such as

$$F(\boldsymbol{x}) = f(\boldsymbol{x}) + \delta \tag{F1.8.5}$$

where δ is a normally distributed random variable with expectation zero and constant variance. This yields

$$E[\![F(\boldsymbol{x})]\!] = E[\![f(\boldsymbol{x})]\!] \tag{F1.8.6}$$

which is easier to deal with than equation (F1.8.4).

The robustness of the strategy depends very much on the selection scheme. It should be obvious that nonelitist schemes are usually superior to elitist variants, since in the latter case outliers can quickly take over the whole population. Furthermore, the robustness with respect to noise can be improved by either increasing the sample size t or enlarging the population size λ. Theoretical and empirical studies (Beyer 1993, Hammel and Bäck 1994) on evolution strategies suggest that it is almost always preferable to increase t except for very small λ. This is not true for the canonical genetic algorithm (Fitzpatrick and Grefenstette 1988, Goldberg *et al* 1993).

Finally, we would like to mention *robust design* as another promising field for stochastic optimization techniques. Here, robust optima with low sensitivity according to small parameter variations are searched for. These variations reflect, e.g. tolerances in a production process, requiring robust parameter settings for technical realizations (Greiner 1994).

F1.8.4 Alternative approaches

Optimization in the context of simulation is an active field of research. We believe that it is almost impossible to give a complete overview on a single page. Thus, we will restrict this subsection to some pointers to the relevant literature.

It is frequently stated that optimization of simulation models can hardly be automated due to the difficulties in the design of a sound objective function, the complexity of the response surface, and resource consumption of single simulation runs. Thus, very often, the exploration of the parameter space is done by hand. Some practitioners prefer a purely random search, sometimes also called 'Monte Carlo simulation', because of its simplicity.

The most common approaches, such as experimental design techniques, metamodeling, response surface methodology, and perturbation methods, just to mention some catchwords, are based on statistics (Law and Kelton 1991, Kleijnen 1992, Pflug 1992), but traditional direct search strategies (Schwefel 1995) may be successfully applied, too.

Very few comparative case studies can be found including evolutionary algorithms (Hampel 1981).

References

Alander J T 1994 *An Indexed Bibliography of Genetic Algorithms: Years 1957–1993* (Espoo: Art of CAD)

Bäck T, Beielstein T, Naujoks B and Heistermann J 1995 Evolutionary algorithms for the optimization of simulation models using PVM *Euro PVM '95, Users Group Meeting (Lyon, 1995)* ed J Dongarra, M Gengler, B Rourancheau and X Vigouroux (Paris: Hermès)

Beyer H-G 1993 Towards a theory of evolution strategies: some asymptotical results from the $(1^{+},\lambda)$-theory *Evolut. Comput.* **1** 165–88

Davis L (ed) 1991 *Handbook of Genetic Algorithms* (New York: Van Nostrand Reinhold)

Ermoliev Y and Wets J-B (ed) 1988 *Numerical Techniques for Stochastic Optimization* (Berlin: Springer)

Fitzpatrick J M and Grefenstette J J 1988 Genetic algorithms in noisy environments *Machine Learning* **3** 101–20

Fonseca C M and Fleming P J 1995 An overview of evolutionary algorithms in multiobjective optimization *Evolut. Comput.* **3** 1–16

Geist G A and Sunderam V S 1992 Network based concurrent computing on the PVM system *J. Concurrency: Practice Experience* **4** 293–311

Goldberg D E, Kalyanmoy D and Clark J H 1993 Accounting for noise in the sizing of populations *Foundations of Genetic Algorithms 2* ed L D Whitley (San Mateo, CA: Morgan Kaufmann)

Greiner H 1994 *Robust Filter Design by Stochastic Optimization, SPIE* vol 2253 (Bellingham, WA: SPIE) pp 150–60

Hammel U and Bäck T 1994 Evolution strategies on noisy functions: how to improve convergence properties *Parallel Problem solving from Nature—PPSN III (Proc. Int. Conf. on Evolutionary Computation and 3rd Conf. on Parallel Problem Solving from Nature, Jerusalem, October 1994)* (*Lecture Notes in Computer Science 866*) ed Yu Davidor, H-P Schwefel and R Männer (Berlin: Springer) pp 159–68

Hampel C 1981 *Ein Vergleich von Optimierungsverfahren für die zeitdiskrete Simulation* PhD Thesis, Technische Universität Berlin

Hahn S, Becks K H and Hemker A 1992 Optimizing Monte Carlo generator parameters using genetic algorithms *New Computing Techniques in Physics Research II, Proc. 2nd Int. Workshop on Software Engineering, Artificial Intelligence and Expert Systems in High Energy and Nuclear Physics (La Londe-les-Maures)* ed D Perret-Gallix (Singapore: World Scientific) pp 267–73

Kall P and Wallace S W 1994 *Stochastic Programming* (Chichester, UK: Wiley)

Kheir N A (ed) 1988 *Systems Modeling and Computer Simulation* (New York: Dekker)
Kleijnen J 1992 *Simulation: a Statistical Perspective* (Chichester, UK: Wiley)
Klir G J 1991 *Facets of Systems Science* (New York: Plenum)
Law A M and Kelton W D 1991 *Simulation Modeling and Analysis* (New York: McGraw-Hill)
Lohmann R 1992 Structure evolution in neural systems *Dynamic, Genetic, and Chaotic Programming* ed B Soucek and the IRIS Group (New York: Wiley) pp 395–411
Michalewicz Z 1994 *Genetic Algorithms + Data Structures = Evolution Programs* (Berlin: Springer)
Pflug G (ed) 1992 Simulation and optimization *Proc. Int. Workshop on Computationally Intensive Methods in Simulation and Optimization, International Institute for Applied Systems Analysis (IIASA) (Laxenburg, 1990)* (Berlin: Springer)
Rechenberg I 1994 Evolutionsstrategie '94 *Werkstatt Bionik und Evolutionstechnik* vol 1 (Stuttgart: Frommann-Holzboog)
Schwefel H-P 1995 *Evolution and Optimum Seeking (Sixth-Generation Computer Technology Series)* (New York: Wiley)

F1.9 Multicriterion decision making

Jeffrey Horn

Abstract

Applying evolutionary computation (EC) to multicriterion decision making addresses two difficult problems: (i) searching intractably large and complex spaces and (ii) deciding among multiple objectives. Both of these problems are open areas of research, but relatively little work has been done on the *combined* problem of searching large spaces to meet multiple objectives. While multicriterion decision analysis usually assumes a small number of alternative solutions to choose from, or an 'easy' (e.g. linear) space to search, research on robust search methods generally assumes some way of aggregating multiple objectives into a single figure of merit. This traditional separation of search and multicriterion decisions allows for two straightforward hybrid strategies: (i) make multicriterion decisions *first*, to aggregate objectives, then apply EC search to optimize the resulting figure of merit, or (ii) conduct multiple EC searches *first* using different aggregations of the objectives in order to obtain a range of alternative solutions, then make a multicriterion decision to choose among the reduced set of solutions. Over the years a number of studies have successfully used one or the other of these two simple hybrid approaches. Recently, however, many studies have implemented Pareto-based EC search to sample the entire Pareto-optimal set of nondominated solutions. A few researchers have further suggested ways of *integrating* multicriterion decision making and EC search, by iteratively using EC search to sample the tradeoff surface while using multicriterion decision making to successively narrow the search. Although all these approaches have received only limited testing and analysis, there are few comparable alternatives to multicriterion EC search (for searching intractably large spaces to meet multiple criteria).

F1.9.1 Introduction

One could argue that real-world problems are, in general, multicriterion. That is, the problems involve multiple objectives to be met or optimized, with the objectives often conflicting. (The terms *objective*, *criterion*, and *attribute* are sometimes subtly distinguished in the literature, but here they are used interchangeably to mean one of a set of goals to be achieved (e.g. cost, to be reduced).) The application of evolutionary search to multicriterion problems seems a logical next step for the evolutionary computation (EC) approaches that have been successful on hard single-criterion problems. Indeed, quite a few EC approaches have found very satisfactory 'tradeoff solutions' in multiobjective problems. However EC search can be, and has been, applied to multiobjective problems in a number of very different ways. It is far from clear which, if any, approaches are superior for general classes of multiobjective problems. At this early point in the development of multicriterion ECs, it would be a good idea to try more than one approach on any given problem.

F1.9.2 Description of the multicriterion application domain

Multicriterion problems in general involve two 'quasiseparable' types of problem difficulty: *search* and *multicriterion decision making*. As with single-criterion problems, the space to be searched can be too large to be enumerated, and too complex (e.g. multimodal, nonlinear) to be solved by linear programming or local or gradient search. In addition to search space complexity, the multiple objectives to be achieved may be conflicting, so that difficult tradeoffs must be made by a rational human decision maker (DM) when ranking solutions. (Indeed, as Goldberg (1989) points out, if our multiple objectives never conflict over the set of feasible solutions, then we do not have any difficulty with the multiple objectives. The search space is then completely (totally) ordered, not just partially ordered, and any monotonic aggregation of the multiple objectives into a single objective will maintain this ordering.)

Traditionally these two aspects of the overall problem, search and multicriterion decision making, are treated separately, and often one or the other is assumed away. Most approaches to searching intractably large spaces (e.g. EC, *simulated annealing* (SA), *tabu search*, stochastic hillclimbing) assume a single objective to be optimized. At the same time, the extensive literature on multiobjective optimization generally assumes a small, enumerable search space, so that the multicriterion decision, not search, is the focus of analysis. EC is in a unique position to address *both* search and multicriterion decisions because of its ability to search partially ordered spaces for multiple alternative tradeoffs. Here we assume that both difficulties are necessarily present, or we would not have a *multicriterion search problem*, suited for multicriterion EC optimization.

D3.5.2, D3.5.4

F1.9.2.1 Practical applications

Multicriterion problems are common. For example, imagine a manufacturing design problem, involving a number of decision variables (e.g. materials, manufacturing processes), and two criteria: manufacturing *cost* and product *quality*. Cost and quality are often conflicting: using more durable materials in the product increases its useful lifetime but increases cost as well. This conflict gives rise to the multicriterion decision problem: what is the optimal tradeoff of cost versus quality? Other possible objectives include lowering *risk* or *uncertainty*, and reducing the number of constraint violations (Richardson *et al* 1989, Liepins *et al* 1990, Krause and Nissen 1995). Another common source of multiple conflicting objectives is the case of multiple decision makers with different preferences, and hence different orderings of the alternatives. Even if each DM could aggregate his or her different criteria into a single ranking of all of the possible alternatives, satisfying *all* of the DMs is a multicriterion problem, with each DM's ranking (rating or ordering) being treated as a separate criterion.

F1.9.2.2 The search space

More formally, we assume a multicriterion problem is characterized by a vector of d decision variables and k criteria. The vector of decision variables can be denoted by $\boldsymbol{X} = (x_0, x_1, x_2, \ldots, x_{d-1})$, just as with any single-objective optimization problem, but in the multiobjective case the evaluation function F is vector valued: $F : \boldsymbol{X} \rightarrow \boldsymbol{A}$, where $\boldsymbol{A} = (a_0, a_2, \ldots, a_{k-1})$ for the k attributes. Thus $F(\boldsymbol{X}) = (f_0(\boldsymbol{X}), f_1(\boldsymbol{X}), \ldots, f_{k-1}(\boldsymbol{X}))$, where $f_i(\boldsymbol{X})$ denotes a function mapping the decision variable vector to the range of the single attribute a_i (e.g. $f_i : \boldsymbol{X} \rightarrow \mathbb{R}$ if a_i is real-valued, and $F : \boldsymbol{X} \rightarrow \mathbb{R}^k$).

Search and multicriterion decisions are not independent tasks. Making some multicriterion choices before search can alter the 'fitness landscape' of the search space by adding more 'ordering information', while search before decision making can eliminate the vast number of inferior (dominated) solutions and focus decision making on a few clear alternatives. Thus the integration of search and multicriterion decision making is a key issue in EC approaches to this application domain, and the type and degree of such integration distinguishes three major categories of multiobjective EC algorithms, below.

F1.9.3 Evolutionary computation approaches

For all of these approaches, the issues of solution representation (i.e. chromosomal encoding), and genetic variation (i.e. the recombination and mutation operators), are the same as for traditional, single-criterion EC applications. There are no special considerations for choosing the encoding or designing the crossover and mutation operators (with the exception of possible mating restrictions for Pareto-based approaches). The major difference in the multicriterion case is in the objective function. The different (i.e. vector-valued)

objective function affects the design of the fitness function and the selection operator, thus these are the EC components we focus on below.

Here we choose to classify approaches according to how they handle the two problems of search and multicriterion decisions. At the highest level, there are three general orderings for conducting search and making multicriterion decisions:

(i) make multicriterion decisions *before* search (decide $\Rightarrow$ search),
(ii) search *before* making multicriterion decisions (search $\Rightarrow$ decide), and
(iii) integrate search and multicriterion decision making (decide $\Leftrightarrow$ search).

F1.9.3.1 Multicriterion decisions before search: aggregation

By far the most common method of handling multiple criteria, with or without EC search, is to *aggregate* the multiple objectives into a single objective, which is then used to totally order the solutions. Aggregative methods can be further divided into the *scalar-aggregative* and the *order-aggregative* (*nonscalar*) approaches.

The scalar-aggregative approach. The most common aggregative methods combine the various objectives into a single scalar-valued *utility function*, $U(\boldsymbol{A})$, where $U : \mathbb{R}^k \rightarrow \mathbb{R}$, reflecting the multicriterion tradeoff preferences of a particular DM. The composite function $U \circ F$ can then be used as the fitness function for EC. A scalar fitness function is required for certain types of selection method, such as fitness-proportionate selection (e.g. roulette wheel, stochastic remainder), although other selection methods require only a complete ordering (e.g. linear ranking) or merely a partial ordering (e.g. tournament selection).

The simplest example of a scalar aggregation is a *linear combination* (i.e. *weighted sum*), such as $U(\boldsymbol{A}) = w_0 a_o + w_1 a_1 + \cdots + w_{k-1} a_{k-1}$, where the w_i are constant coefficients (i.e. weights). The DM sets the weights to try to account for his or her relative ratings of the attributes. For example, Bhanu and Lee (1994, chs 4, 8) sum five measures of image segmentation quality into a single objective using equal weights, while Vemuri and Cedeño (1995) first rank the population k times using each of the criteria, then for each solution sum the k criterion rankings, rather than the attribute values themselves. One drawback of the linear combination approach is that it can only account for linear relationships among the criteria. It can be generalized to handle nonlinearities by introducing nonlinear terms, such as exponentiating critical attributes, or multiplying together pairs of highly dependent attributes. One can introduce such nonlinear terms in an *ad hoc* manner, guided only by intuition and trial and error, but we restrict our discussion to more systematic and generalizable methods below.

A very common *nonlinear* scalar aggregation is the *constraint* approach. Constraints can handle nonlinearities that arise when a DM has certain *thresholds* for criteria, that is, maximum or minimum values. For example, a DM might be willing to sacrifice quality to save money, but only down to a certain level. Typically, when a solution fails to meet a constraint, its utility is given a large *penalty*, such as having a large fixed value subtracted or divided into the total score (see e.g. Simpson *et al* 1994, Savic and Walters 1995, Krause and Nissen 1995). Richardson *et al* (1989) give guidelines for using penalty functions with GAs, while Stanley and Mudge (1995) discuss turning constraints 'back into' objectives.

C5
C5.2

A recent development in the decision analysis community handles multiplicative nonlinearities: *multiattribute utility analysis (MAUA)* (Keeney and Raiffa 1976, de Neufville 1990, Horn and Nafpliotis 1993). Under MAUA, separate utility functions for each attribute, $u_i(a_i)$, are determined for a particular DM in a systematic way, and incorporate attitude toward uncertainty (in each single attribute). These individual utility functions are then combined by multiplication (rather than addition). Through a series of lottery-based questions, the DM's pairwise tradeoffs (between pairs of attributes) are estimated, and incorporated into the coefficients (weights) for the multiplicative terms.

A similar multiplicative aggregation is used by Wallace *et al* (1994). They first determine a DM's *probability of acceptance* function for each criterion, to take into account nonlinear attitudes toward individual criteria (e.g. thresholds). The acceptance probability functions are then multiplied together, giving the overall probability of acceptance, and the logarithm taken (to reduce selection pressure).

Another nonlinear scalar aggregative method is the *distance-to-target* approach. A target attribute vector $\boldsymbol{T}$ is chosen as an ideal solution to shoot for. Solutions are evaluated by simply measuring their distance from this theoretical goal in criterion space (see e.g. Wienke *et al* 1992). Choosing the target vector and the form of the metric both involve multicriterion decisions by the DM. In the case of the metric,

the scaling of the attributes greatly affects the relative distances to the goal, while the actual formula for the metric can also change the ordering of solutions. Consider the general class of *Holder* metrics,

$$h_p(\boldsymbol{A}, \boldsymbol{B}) = \left(\sum_{i=0}^{k-1} |a_i - b_i|^p \right)^{1/p} \qquad p \geq 1. \tag{F1.9.1}$$

For example, $p = 1$, which is known as the *metropolitan* or 'city block' metric, gives a linear combination of attributes, while the more common $p = 2$ Euclidean distance introduces nonlinearities. In general, increasing the order p of the Holder norm increases the degree of nonlinear interaction among attributes.

More sophisticated refinements of the simple target distance measure include the *technique for order preference by similarity to ideal solution* (TOPSIS) approach, which seeks to minimize the distance to a 'positive ideal solution' while simultaneously maximizing the distance from a 'negative ideal solution'. (Thus TOPSIS attempts to reduce a k-criterion problem to a $k = 2$, bicriterion one.) Hwang *et al* (1993) use TOPSIS to aggregate multiple objectives for GA optimization.

A significant variant of the distance-to-target approach is the *minimax*, or *MinMax*, formulation (Osyczka 1984, Srinivas, and Deb 1995). Minimax seeks to minimize the maximum 'criterion distance' to the target solution $\boldsymbol{T}$. Choosing the maximum of the k criterion distances is equivalent to using the *maximum Holder metric*, which is obtained as $p \to \infty$ in equation (F1.9.1) above: $h_\infty(\boldsymbol{A}, \boldsymbol{T}) = \max(|a_0 - t_0|, |a_1 - t_1|, \ldots, |a_{k-1} - t_{k-1}|)$. Minimizing this distance becomes the single objective. By differentially scaling the individual criterion differences in the minimax calculation, we obtain *Tschebycheff's weighting method* (Steuer 1986), which, unlike linear aggregations, can be used to sample concave portions of the Pareto-optimal frontier (Cieniawski 1993) (see 'Independent sampling' in section F1.9.3.2 below).

The order-aggregative approach. Not all aggregations are scalar. For example, the *lexicographic approach* gives a total ordering of all solutions (thus they can be ranked from best to worst), without assigning scalar values. The approach requires the DM to order the criteria. Solutions are then ranked by considering each attribute in order. As in a dictionary, lexicographic ordering first orders items by their most important attribute. If this results in a tie, then the second attribute is considered, and so on. Fourman (1985) uses a lexicographic ordering when comparing individuals under *tournament selection* C2.3 in a GA. Nonscalar, order-aggregative methods (e.g. lexicographic ordering or voting schemes) impose a total ordering on the space of solutions, but do not provide any meaningful scalar evaluation useful for a fitness function. Therefore such methods are best suited for rank-based selection, including tournament selection.

F1.9.3.2 Search before multicriterion decisions: seeking the Pareto frontier

The aggregative approaches are open to the criticism of being overly simplistic. Is it possible to combine the conflicting objectives into a single preference system, prior to search? Or are some criteria truly noncommensurate? Recognizing the difficulty, perhaps the impossibility, of making all of the multicriterion decisions up front, many users and researchers have chosen to first apply search to find a set of 'best alternatives'. Multicriterion decision-making methods can then be applied to the reduced set of solutions.

Vilfredo Pareto (1896) recognized that, even without making any multicriterion decisions, the solution space is already *partially ordered*. Simply stated, the *Pareto criterion* for one solution to be superior to B2.1.3 another is for it to be at least as good in all attributes, and superior in at least one. More formally, given k attributes, all of which are to be maximized, a solution A, with attribute values $(a_0, a_1, a_2, \ldots, a_{k-1})$, and a solution B, with attribute values $(b_0, b_1, b_2, \ldots, b_{k-1})$, we say that A *dominates* B if and only if $\forall i : a_i \geq b_i$, and $\exists j : a_j > b_j$. The binary relation of dominance partially orders the space of alternatives. Some pairs of solutions will be *incomparable*, in that neither dominates the other (since one solution might be better than the other in some attributes, and worse in others). Clearly this partial ordering will be agreed to by all rational DMs. Therefore, all dominated solutions can be eliminated from consideration before the multicriterion decisions are made. In particular, the set of solutions not dominated by any solution in the entire space is desirable in that such a set *must* contain all of the possible optimal solutions according to *any* rational DM's multicriterion decisions. This *nondominated set* is known by many names: the *Pareto-optimal set*, the *admissible set*, the *efficient points*, the *nondominated frontier*, and the *Pareto front*, for example.

The terms *front* and *frontier* arise from the geometric depiction of the *criterion space* (or *attribute space*). Figure F1.9.1 depicts a $k = 2$-criterion space. Here the chosen criteria are cost (to be minimized)

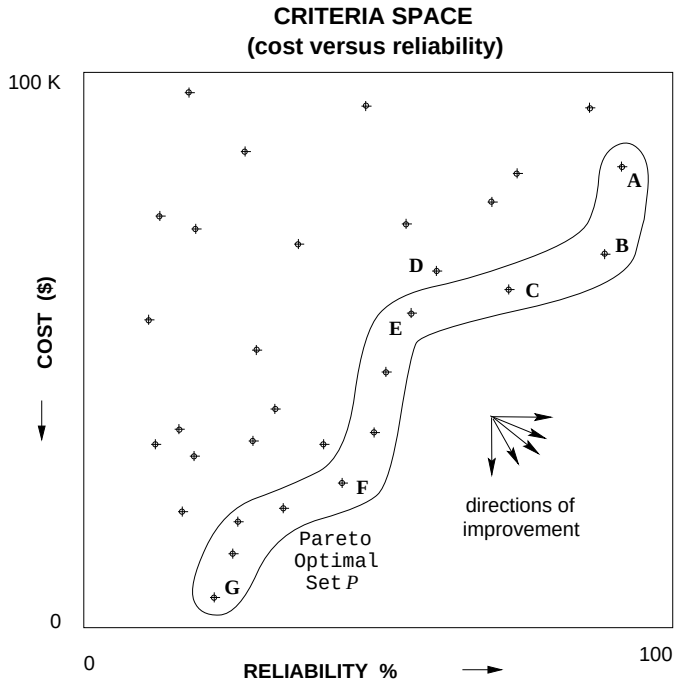


Figure F1.9.1. Criterion space in an example two-criterion problem.

and reliability (to be maximized). A population of individuals (candidate solutions) is plotted using each individual's evaluated criterion vector as a coordinate. Note that in figure F1.9.1 individual C dominates D, but does not dominate E (they are incomparable). The set of all individuals not dominated by any member of the population (i.e. P) is circled.

The Pareto-optimal set P is desirable as input to the multicriterion decision-making process for several reasons. (i) Knowledge of the nature of P might simplify the multicriterion decision. For example, P might be singular ($|P| = 1$), with one solution dominating the rest. Or P might at least be small enough to allow a DM, or a team of DMs, to consider all choices at once, in detail. Even if P is large, one solution might stand out, such as an extremum (e.g. A or G in figure F1.9.1), or perhaps a 'knee' of the front (e.g. B or F in figure F1.9.1), at which large sacrifices in one attribute yield only small improvements in the other(s). (ii) P is 'DM independent'. If a DM (or his or her preferences) changes, the Pareto search need not be performed again. (iii) Interpolating a smooth curve through the samples (P) of the front, although potentially misleading, can give some idea of how the attributes interact, and so focus subsequent search on poorly sampled but promising directions (Fonseca and Fleming 1993a, b, Horn and Nafpliotis 1993). Thus, Pareto approaches allow the study of 'tradeoffs', not just solutions.

All of the approaches described in this section seek the Pareto-optimal set (although some authors do not mention Pareto optimality and instead talk of simply finding multiple good tradeoffs). On the notation P: All of the methods below attempt to evolve a population toward the actual Pareto frontier; we call this P_{actual}. Any given population (e.g. generation) has a nondominated subset of individuals, $P_{\text{on-line}}$. The hope is that by the end of the run, $P_{\text{on-line}} = P_{\text{actual}}$, or at least $P_{\text{on-line}} \subset P_{\text{actual}}$. (Of course in an open problem we generally have no way of knowing P_{actual}.) In addition, it is generally assumed that any practical implementations of the algorithms will maintain off-line a set $P_{\text{off-line}}$ of the best (nondominated) solutions found during the run so far, since stochastic selection methods might cause the loss of nondominated solutions. $P_{\text{off-line}}$ thus represents the nondominated set of all solutions generated so far during a run. Some algorithms use elitism to ensure that $P_{\text{on-line}} = P_{\text{off-line}}$, while others occasionally insert members of $P_{\text{off-line}}$ back into $P_{\text{on-line}}$ during the run.

Independent sampling (multiple single-criterion searches). One straightforward approach to finding members of P_{actual} is to use multiple single-criterion searches to optimize different aggregations of the criteria. For example, we could try optimizing one criterion at a time. (If successful, this would give us the extrema (corners) of the Pareto-optimal tradeoff surface, e.g. individuals A and G in figure F1.9.1.)

Alternatively we could assume a linear combination (weighted sum) of the objectives, and vary the weights from search to search to gradually build up a sampling of the front.

Fourman (1985), one of the first to perform independent sampling, reports the use of several composite formulae to sample the tradeoff surface. These include linear combinations and lexicographic orderings. In each case Fourman varies the exact formula systematically to obtain different tradeoffs.

Ritzel *et al* (1994) discuss multiple GA runs to optimize one criterion at a time, while holding the other criteria constant (using constraints). They then vary the constraint constants to obtain the entire tradeoff surface.

Cieniawski (1993) runs a single-objective GA several times, using the fitness function $F(\boldsymbol{X}) = f_1(\boldsymbol{X}) + \alpha f_2(\boldsymbol{X})$ (where f_1 and f_2 are the two criterion functions). He gradually increases α from zero. Similarly, Tsoi *et al* (1995) and Chang *et al* (1995) both apply a single-criterion GA to optimize $F(\boldsymbol{x}) = \beta f_1(\boldsymbol{x}) + (1-\beta) f_2(\boldsymbol{X})$, varying β from zero to one in equal increments, to build up a picture of a two-dimensional tradeoff surface of f_1 versus f_2. Note that the number of sample points needed to maintain a constant sampling density increases exponentially in k.

Linear aggregative methods, however, are biased toward convex portions of the tradeoff curve. No linear combination exists that will favor points in the concave portions as global optima. For example, solution E in figure F1.9.1 will be inferior to some other member of P no matter what weights are used in the summation. However a *nonlinear* aggregation can be used to sample concave portions. For example, Cieniawski runs multiple single-objective GA searches using Tschebycheff's weighting method (discussed above) on a bicriterion problem: $F(\boldsymbol{X}) = \max[(1-\beta)|f_1(\boldsymbol{T}) - f_1(\boldsymbol{X})|, \beta|f_2(\boldsymbol{T}) - f_2(\boldsymbol{X})|]$, varying β from zero to one by 0.05 to obtain both convex and concave portions of the Pareto-optimal frontier.

Cooperative population searches. Rather than conduct multiple independent single-objective searches, many recent studies have implemented a simultaneous parallel search for multiple members of P_{actual} using a single large population in the hope that the increased implicitly parallel processing (of schemata) will be more efficient and effective. Again, there are several ways to do this, including *criterion selection*, *aggregation selection*, and *Pareto selection.*

Cooperative population searches (with criterion selection). Three independent studies (Schaffer 1984, 1985, Fourman 1985, Kursawe 1990, 1991) all implement the same basic idea: parallel single-criterion search, or *criterion selection*, in which fractions of the next generation are selected according to one of the k criteria at a time. Probably the first such criterion selection study, and probably also the first multicriterion population-based search in general, was that implemented by Schaffer (1984, 1985), using his *vector-evaluated genetic algorithm* (VEGA). VEGA selects a fraction $1/k$ of each new population (next generation) using one of each of the k attributes. (Crossover and mutation are applied to the entire population.) VEGA demonstrated for the first time the successful use of the GA to find multiple members of P using a single population (see e.g. Schaffer and Grefenstette 1985). Fourman (1985), disappointed with the performance of weighted sum and lexicographic ordering aggregations for multiple independent sampling, proposed a selection method similar to VEGA's. Fourman conducts binary tournaments, randomly choosing one criterion to decide each tournament. Later, Kursawe (1990, 1991) implemented a randomized criterion selection scheme almost identical to Fourman's. Kursawe suggests that the criterion probabilities be completely random, fixed by the user, or allowed to evolve with the population. He adds a form of *crowding* (De Jong 1975), as well as dominance and diploidy (Goldberg 1989), to maintain C6.1.3 Pareto diversity. These three similar criterion selection approaches have all been subject to some criticism for being potentially biased against 'middling' individuals (i.e. those solutions not excelling at any one particular objective) (Richardson *et al* 1989, Goldberg 1989, and, empirically, Murata and Ishibuchi 1995, Krause and Nissen 1995, Ishibuchi and Murata 1996).

Cooperative population searches (with aggregation selection). In an attempt to promote more of the 'middling' individuals than do the above criterion selection methods, Murata and Ishibuchi (1995, Ishibuchi and Murata 1996) claim to generalize Kursawe's algorithm. Rather than just use random criteria for selection, their *multiobjective GA* (MOGA) uses random linear combinations of criteria. That is, they randomly vary the weights, $w_i \in [0, 1]$ in the summed fitness function $F(\boldsymbol{X}) = \sum_{i=0}^{k-1} w_i f_i(\boldsymbol{X})$.

Cooperative population searches (with Pareto selection). Since 1989, several studies have tried to remain truer to the Pareto criterion by using some form of explicit Pareto selection, such that selection favors Pareto-optimal solutions (that is, members of $P_{\text{on-line}}$) above all others, and no preferences are given *within* the Pareto-optimal ($P_{\text{on-line}}$) equivalence class. Many of these efforts have incorporated some form of active *diversity promotion*, such as GA *niching*, to find and maintain an even distribution (sampling) of points along the Pareto front. C6.1

Pareto ranking. Goldberg (1989) describes *nondominated sorting* to rank the population according to Pareto optimality. Under such selection, the currently nondominated individuals in the population are given rank one and then removed from the population. The newly nondominated individuals in the reduced population are assigned rank two, and removed. This process continues until all members of the original population are ranked. Goldberg also suggested the use of niching and speciation methods to promote and maintain multiple subpopulations along the Pareto optimal front, but he did not recommend a particular niching method. Goldberg did not implement any of these suggestions at that time.

Hilliard *et al* (1989) implement Goldberg's nondominated sorting, but without niching. Their *Pareto GA* applies proportionate selection to the nondomination ranks. Liepins, *et al* (1990) also implement Goldberg's nondominated sorting, without niching, but use Baker's (1985) method of rank-based selection (applied to the nondominated sorting ranks). More recently, Ritzel *et al* (1994) have implemented Goldberg's nondominated sorting in their *Pareto GA*, again without niching. They conduct binary tournament selection using the ranks for comparison.

Pareto ranking plus niching (fitness sharing). In 1993, four groups independently implemented Goldberg's suggestions for combined Pareto selection and niching (using the *fitness sharing* of Goldberg and Richardson (1987)), but in different ways: MOGA, NPGA, NSGA, and the Pareto-optimal ranking GA with sharing. C6.1.2

The *multiobjective GA* (MOGA) of Fonseca and Fleming (1993a, b, 1994, 1995b–d) ranks the population according to the 'degree of domination': the more members of the current population that dominate a particular individual, the lower its rank. This ranking is finer grained than Goldberg's, in that the former can distinguish more ranks than the latter. (Note that any method of ranking a partially ordered set allows the use of traditional rank-based selection schemes on Pareto-ordered spaces.) Apparently, fitness sharing (Goldberg and Richardson 1987) takes place within each rank only, such that members within each Pareto rank are further ranked according to their fitness sharing *niche counts*. (A niche count is a measure of how crowded the immediate 'neighborhood' of an individual is. The more close neighbors, the higher the niche count.) Fonseca and Fleming measure distance (for niche counting) in the *criterion space* (or *attribute space*). Recently, Shaw and Fleming (1996a) have applied the Pareto ranking scheme of Fonseca and Fleming to a $k = 3$-criterion scheduling problem, both with and without niching (and mating restrictions).

Rather than ranking, the *niched Pareto GA* (NPGA) of Horn and Nafpliotis (1993, Horn *et al* 1994) implements *Pareto domination tournaments*, binary tournaments using a sample of the current population to determine the dominance status of two competitors A and B. If one of the competitors is dominated by a member of the sample, and the other competitor is not dominated at all, then the nondominated individual wins the tournament. If both or neither are dominated, then fitness sharing is used to determine the winner (i.e. whichever has the lower niche count). The sample size (t_{dom}) is used to control 'Pareto selection pressure' analogously to the use of tournament size in normal (single-objective) tournament selection. The Pareto domination tournament can be seen as a locally calculated, stochastic approximation to the globally calculated degree-of-domination ranking of Fonseca and Fleming.

The *non-dominated sorting GA* (NSGA) of Srinivas and Deb (Srinivas 1994, Srinivas and Deb 1995), implements Goldberg's original suggestions as much as possible. NSGA uses Goldberg's suggested Pareto ranking procedure, and incorporates fitness sharing. Unlike MOGA and NPGA, NSGA performs sharing in the phenotypic space (rather than the criterion space), calculating distances between decision variable vectors. Michielssen and Weile (1995) recently combined the nondominated sorting selection of NSGA with the criterion space sharing of MOGA and NPGA.

Eheart *et al* (1993) and Cieniawski *et al* (1985) apply Goldberg's nondominated sorting to rank the population, as in NSGA, but then use the ranks as objective fitnesses to be degraded by sharing. (Note that this is different from MOGA, NPGA, and NSGA, which attempt to limit the effects of sharing to

competition *within* ranks, not between.) They then apply tournament selection, using the shared fitnesses, but do not use the 'standard' fitness sharing of Goldberg and Richardson (1987). Instead of dividing the objective fitness by the niche count, they add the niche count to the rank. Although they perform sharing in criterion space, as in MOGA and NPGA, they measure distance (i.e. similarity between pairs of individuals) along only one dimension (e.g. 'reliability', in their 'cost' versus 'reliability' bicriterion problem). This biases diversity toward the chosen criterion.

The four efforts above, inspired by Goldberg's 1989 suggestions and incorporating fitness sharing to promote niching within Pareto-optimal equivalence classes, are not the only explicitly Pareto selective approaches in the literature. Some of the alternative Pareto selection schemes below implicitly or explicitly maintain at least some diversity in the P-optimal set without using fitness sharing.

Pareto elitist recombination. Louis and Rawlins (1993) hold four-way Pareto tournaments among two parents and their two (recombined and mutated) offspring. Such a tournament can be seen as the generalization, to multiple objectives, of the *elitist recombination* of Thierens and Goldberg (1994) for single-objective GAs. The parent–offspring replacement scheme should result in some form of quasistable niching (Thierens 1995). This 'parent–offspring nondomination tournament' is applied by Gero and Louis (1995) to beam shape optimization, and generalized to μ parents and λ offspring in a $(\mu + \lambda)$-ES by Krause and Nissen (1995).

Simple Pareto tournaments with demes. Poloni (1995) and Baita *et al* (1995) hold binary Pareto tournaments, in which an individual that dominates its competitor wins. If neither competitor dominates, a winner is chosen at random. Poloni uses a distributed GA, with multiple small populations, or *demes*, relatively isolated (i.e. little or no migration), as a niching method to try to maintain Pareto diversity. Langdon (1995) generalizes the simple binary Pareto tournament to any number $m \geq 2$ of competitors. A single winner is randomly chosen from the Pareto-optimal subset of the m randomly chosen competitors. Langdon favors a steady-state GA, using m-ary Pareto tournaments for deletion selection as well. He also uses demes as a means to maintain diversity. Later, Langdon (1996) adds a generalization of the Pareto domination tournaments of Horn and Nafpliotis (1993): if none of the m randomly chosen competitors dominates all of the others, then a separate random sample of the population is used to rank the m competitors. The competitor that is 'least dominated' (i.e. dominated by the fewest members of the sample) wins, a stochastic approximation to the degree-of-dominance Pareto ranking of Fonseca and Fleming (1993a). Langdon points out correctly that such sampled domination tournaments induce a niching pressure, although he apparently continues to use demes as well (Langdon 1996).

Pareto elitist selection. Some researchers recently have implemented *Pareto elitist selection* strategies. These approaches divide the population into just two ranks: dominated and nondominated (i.e. $P_{\text{on-line}}$ and non-$P_{\text{on-line}}$). Although strongly promoting $P_{\text{on-line}}$, these algorithms differ in the extent to which they preserve such individuals from one generation to the next. For example, Belegundu *et al* (1994) select only rank one (i.e. $P_{\text{on-line}}$) for reproduction. (Random individuals are generated to maintain the fixed population size N.) Tamaki *et al* (1995) propose a somewhat less severe *Pareto reservation strategy* that copies all nondominated individuals into the next generation's population. If additional individuals are needed to maintain the fixed population size N, they are selected from the dominated set using criterion selection. Similarly, if $|P_{\text{on-line}}| > N$, then individuals are deleted from the population (i.e. from $P_{\text{on-line}}$) using 'criterion deletion'. (Tamaki *et al* (1996) have recently added fitness sharing (in the criterion space) to the Pareto reservation strategy to promote explicitly Pareto diversity (i.e. diversity within $P_{\text{on-line}}$).) Applications of their approach can be found in Tamaki *et al* (1995) and Yoshida *et al* (1996).

Other Pareto elitist selection methods include the *Pareto-optimal selection method* of Takada *et al* (1996). According to Tamaki *et al* (1996), Takada *et al* apply recombination and mutation first, to generate an intermediate population, then select only the Pareto-optimal set from among the old and intermediate populations (i.e. all of the parents and offspring). Krause and Nissen (1995) implement the same '$(\mu+\lambda)$ Pareto elitism selection' as Takada *et al* but use an ES rather than a GA. Eheart *et al* (1993) and Cieniawski (1993) use a stochastic approximation to Pareto elitist selection, by maintaining $P_{\text{off-line}}$ and constantly reinjecting these individuals back into the populations through random replacements (what they call *Pareto-optimal reinjection*). Apparently, they combine this elitism with Pareto-optimal rank-based tournament selection, with and without niching.

A very unusual Pareto elitist selection method is the *distance method* of Osyczka and Kundu (1995) (applied by Kundu and Kawata (1996) and by Kundu *et al* (1996)). The fitness of an individual is a function of its distance from the current Pareto set $P_{\text{off-line}}$ (as opposed to its distance from an ideal, target vector; see above). This distance is measured in criterion space, using the Euclidean metric, and applying a minimum-distance criterion (i.e. the distance from $\boldsymbol{X}$ to $P_{\text{off-line}}$ is equal to the *minimum* of the distances from $\boldsymbol{X}$ to any member of $P_{\text{off-line}}$). For solutions dominated by $P_{\text{off-line}}$, this distance is negative; otherwise, it is positive. The (signed) distance is then added to the individual's fitness. By incorporating this 'distance to the nearest member of $P_{\text{off-line}}$' into the fitness calculation, the algorithm explicitly promotes criterion-space diversity, both on and off the Pareto frontier, in a manner akin to niching. (Osyczka and Kundu (1996) modify their original method to favor members of $P_{\text{on-line}}$ that dominate members of $P_{\text{off-line}}$, to focus search on rapidly improving portions of the tradeoff surface.)

F1.9.3.3 Integrated search and decision making

A more integrated hybrid of EC search and multicriterion decision making calls for iterative search and decision making. Preliminary multicriterion search is performed to give the DM some idea of the range of tradeoffs possible. The DM then makes some multicriterion decisions to reduce the search space. Additional EC search is limited to this particular region of the criterion space. The iterative process of EC search, multicriterion decisions, EC search, and so on continues until a single solution is left.

Several researchers have suggested such iterative integrations (Horn and Nafpliotis 1993, Poloni 1995), but Fonseca and Fleming (1993a) actually implement one: the *goal attainment method*, an extension of their MOGA. In their approach, the original MOGA is run for a few generations, then the DM considers the current $P_{\text{off-line}}$ and (as above) chooses a target tradeoff point to focus subsequent search. The MOGA is then run for a few more generations using a 'modified multiobjective ranking scheme' that considers both Pareto domination and 'goal attainment' (e.g. distance to target). Fonseca and Fleming briefly discuss the role of the MOGA as a 'method for progressive articulation of [DM] preferences'.

F1.9.3.4 State of the art

Multicriterion EC research has broadened from the early aggregative approaches, with the introduction of criterion selection (e.g. VEGA) in the mid-1980s, the addition of Pareto ranking at the turn of the decade, the combinations of niching and Pareto selection in 1993, and the implementation of many radically different alternative approaches along the way. Two recent reviews (Fonseca and Fleming 1995a, Tamaki *et al* 1996) compare some of the latest algorithms with some of the 'classics'. (In particular, Tamaki *et al* survey several new efforts in Japan which might be inaccessible to non-Japanese readers.) Here we restrict our discussion to a recent trend: hybridization of previously distinct approaches to multicriterion EC optimization.

Most recently (1995, 1996), the EC conferences include a substantial number of *hybrid* methods, combining old and new techniques for dealing with multiple criteria during EC search. We mention a few example hybrids below.

Hybrid ordering: aggregative and Pareto approaches. Stanley and Mudge (1995) combine Pareto and order aggregative approaches to achieve a very fine-grained ranking. They use a DM's strict ordering of the criteria (assuming one exists) to lexicographically order (and rank) individuals *within* each of the ranks produced by Goldberg's nondominated sorting. More recently, Greenwood *et al* (1997) relax the need for a strict ordering of the criteria to permit the DM to perform only an 'imprecise ranking of attributes'. Fonseca and Fleming (1993a) also use an aggregative method (*goal attainment*, a version of the basic distance-to-target approach) in their MOGA with *progressive articulation of preferences*, to further order the solutions within each of the ranks produced by their degree-of-dominance Pareto ranking scheme discussed earlier. Bhanu and Lee (1994, ch 9) use the linear combination method (i.e. weighted sum) to reduce a $k = 5$-criterion image segmentation problem to a $k = 2$ bicriterion problem. They sum three quality measures into a single 'local quality measure', and the other two quality measures into a 'global quality measure'. They then apply criterion selection with Pareto elitism. Tsoi *et al* (1995) also use weighted sums to combine m pollutant emission levels into a single emissions objective. They optimize the bicriterion problem of cost versus total emissions by applying multiple single-criterion GA runs. Fonseca and Fleming (1994) turn three of five objectives into constraints, and apply their Pareto

selective MOGA to find the tradeoff surface of the other two criteria (see figure 5 of Fonseca and Fleming 1994).

Hybrid selection: VEGA and Pareto approaches. One of Cieniawski's (1993) four multiobjective GA formulations is a *combination VEGA and Pareto-optimal ranking GA*. It is hoped that the combination can find criterion specialists (extremes of the Pareto frontier) using criterion selection, and also favor 'middling individuals' in between, using Goldberg's nondominated sorting. To achieve this, Cieniawski simply uses criterion selection for g generations, then switches to the Pareto-ranked tournament selection (using Goldberg's nondominated sorting), without niching. He finds that the combination outperforms VEGA and Pareto ranking by themselves, but is similar in performance to Pareto ranking with fitness sharing.

Tamaki *et al* (1995) also try to balance VEGA's supposed bias towards Pareto extrema by adding Pareto elitism. Their *noninferiority preservation strategy* preserves Pareto-optimal individuals $P_{\text{on-line}}$ from one generation to the next, unless $|P_{\text{on-line}}| > N$, in which case criterion selection is applied to $P_{\text{on-line}}$ to choose exactly N individuals.

In their COMOGA method (constrained optimization by multiobjective genetic algorithms), Surrey *et al* (1995) use criterion selection on two criteria: *cost* and *constraints*. When selecting according to a particular criterion, COMOGA implements the Pareto ranking of Fonseca and Fleming (1993a) (e.g. *constraints* consists of multiple constraints as 'subcriteria').

Hybrid search algorithms. As with single-criterion EC algorithms, many implementors of multicriterion ECs combine deterministic optimizers (e.g. steepest-ascent hillclimbing) or stochastic optimizers (e.g. simulated annealing (SA)) with GA search. Ishibuchi and Murata (1996) add local search to the global selection recombination operators, using their randomly weighted aggregations of attributes (Murata and Ishibuchi 1995) to hillclimb in multiple directions in criterion space, simultaneously. Tamaki *et al* (1995, 1996) also add local search to their hybrid Pareto reservation strategy with niching and parallel (criterion) selection, applying hillclimbing to each member of the population every 100 generations. Poloni (1995) suggests the use of $P_{\text{off-line}}$, found by his Pareto selective GA, by the DM to choose a set of criterion weights and a starting point for subsequent optimization by a domain-specific algorithm (the Powell method).

Tsoi *et al* (1995) create two GA–SA hybrids (essentially GAs with 'cooled' nondeterministic tournament selection) for multiple single-criterion runs to sample P_{actual}. According to Tamaki *et al* (1996), Kita *et al* (1996) apply the SA-like *thermodynamic GA* (TDGA) of Mori *et al* (1995) to a Pareto ranking of the population. The TDGA is designed to maintain a certain level of population 'entropy' (i.e. diversity). Combined with Pareto ranking, the TDGA reportedly can maintain Pareto diversity (i.e. large $P_{\text{on-line}}$).

Hybrid techniques

(i) *Fuzzy evolutionary optimization.* Li *et al* (1996) model the uncertainty in the attribute values using fuzzy numbers. They use a GA to optimize a linear combination of the 'defuzzified' rankings. Their *registered Pareto-optimal solution strategy* simply maintains $P_{\text{off-line}}$, which is *not* used in selection. Li *et al* apply TOPSIS (discussed earlier) to select a 'best' member of $P_{\text{off-line}}$. D2

(ii) *Expert Systems.* Fonseca and Fleming (1993a) suggest the use of an automated DM to interact with an interactive multicriterion EC algorithm, using built-in knowledge of a human DM's preferences to guide EC search. An automated DM would make multicriterion decisions on the fly, based on both its DM knowledge base and on accumulating knowledge of the search space and tradeoff surface from the EC algorithm.

F1.9.4 Comparisons: advantages and disadvantages

From the range of new and different EC approaches mentioned here, it seems too early to expect rigorous, comprehensive performance comparisons, or even the beginnings of a broad theory of multicriterion EC. More research effort is being spent designing new and hybrid multicriterion EC methods than is going into theory development or controlled experimentation.

Direct, empirical comparisons. Most papers on multicriterion EC introduce a new algorithm and compare it with one or two other, well-known approaches (typically VEGA) on a few artificial test problems (e.g. Schaffer's (1985) F2) and one or two open, real-world applications. For example, Hilliard *et al* (1989) compare nondominated sorting to VEGA and to random search, while Tamaki *et al* (1996) compare VEGA, NPGA, MOGA (Fonseca and Fleming), and their own Pareto reservation strategy.

Analytical comparisons. Aside from the scant empirical evidence, we have very little theoretical guidance. We do have conjectures and intuitions, but these sometimes lead to contradictory advice. For example, suppose we want to find P_{actual}. It would seem that a single large cooperative population search for the Pareto front offers greater potential for implicitly parallel, robust search than do multiple independent searches using aggregations of criteria. On the other hand, multiple independent searches might benefit from the concentration of the population in one particular direction in criterion space. In terms of the goal of the EC search, the Pareto goal might at first seem superior to an aggregative goal, since P_{actual} contains the global optima of all monotonic aggregations of the criteria. However if $|P_{\text{actual}}| \gg N$, or if P_{actual} is extremely difficult to find, then the Pareto search might fail, while the aggregative search might find just that one member of P_{actual} that is desired. Then again, even if we can determine prior to search a single aggregation of the criteria that totally orders the search space in complete agreement with the DM's preferences, we might still find that a nonaggregative (e.g. Pareto) search discovers a better optimum than does a single-objective search, because of the additional 'Pareto diversity' in the population.

We are only beginning to understand how single-objective EC algorithms handle nonlinear interactions among decision variables during search. In the multicriterion case, we or our EC algorithms must also deal with nonlinear interactions among criteria. How we choose to handle criterion interactions can change the shape of the entire search space (and vice versa).

Observations. Despite the paucity of empirical comparisons and theory, it seems appropriate to make a few tentative observations based on the current survey of multicriterion EC approaches.

(i) Even if the DM agrees strongly with a particular aggregation of criteria prior to a single-objective EC search, an additional, Pareto search for the tradeoff surface should be conducted anyway. It might yield a better solution than the single-objective search, or it might lead to a reevaluation of the multicriterion decision made earlier.

(ii) Most Pareto EC methods, by using $P_{\text{on-line}}$ to check for dominance, induce a selective pressure toward diversity (i.e. an implicit niching effect), but at least for 'pure Pareto' approaches (e.g. nondominated sorting) this implicit diversity pressure does not exist *within* the $P_{\text{on-line}}$. Without explicit niching, genetic drift takes place within $P_{\text{on-line}}$. Thus the need for an explicit niching mechanism, such as fitness sharing or crowding, depends on the extent of the implicit diversity pressure. Strong Pareto elitism can lead to a mostly nondominated population and hence genetic drift, for example. There seems to be a need to balance *domination* (selective) pressure with *niching* (diversity) pressure (Horn and Nafpliotis 1993).

(iii) Although the Pareto criterion avoids combining or comparing attributes, Pareto EC algorithms, because of their finite populations, *must* make such comparisons. For example, methods that calculate distance in criterion space (e.g. distance to target, distance to $P_{\text{off-line}}$, or pairwise distances for niche counts in fitness sharing) combine distances along different attribute dimensions, and thus are highly sensitive to the relative scaling of the attributes (see e.g. Fonseca and Fleming 1993a and Horn *et al* 1994) for attempts to 'normalize' attribute ranges for sharing distance calculations). However any Pareto EC method in general must allocate a finite population along a dense (potentially infinite) Pareto front, thereby choosing particular directions in criterion space on which to focus search. Again, the scaling of the attributes dramatically affects the shape of the current Pareto front as well as the angular difference between search vectors (e.g. exponentially scaling an attribute can change a portion of P_{actual} from convex to concave (Cieniawski 1993)).

(iv) Criterion selection (e.g. VEGA) and Pareto GAs might be complementary, since the former seem relatively effective at finding extrema of P_{actual} (i.e. solutions that excel at a single criterion), while the latter find many 'middling' (compromise) tradeoffs, yet often fail to find or maintain the best 'ends' of the Pareto-optimal frontier (e.g. Krause and Nissen 1995).

(v) It is far from clear how well any of the EC approaches scale with the number of criteria. Most applications and proof-of-principle tests of new algorithms use two objectives (a few exceptions use

up to seven), but what of much higher-order multicriterion problems? As Fonseca and Fleming (1995a) point out, in general more conflicting objectives mean a flatter partial order, larger P_{actual}, $P_{\text{on-line}}$, and $P_{\text{off-line}}$, and less Pareto selective pressure. Nonaggregative EC approaches, and Pareto EC methods in particular, might not scale up well. Hybrid approaches that aggregate *some* of the criteria (above) might help.

(vi) One major disadvantage of all EC algorithms (or of any stochastic optimizer), compared to enumeration or some other deterministic algorithm, is the same disadvantage nondeterministic search suffers in the single-criterion case: we have no way of knowing when to stop searching, since we have no way of knowing whether our solution is optimal. In the case of the Pareto approaches this weakness becomes acute. A single solution far off the estimated front can dominate a large portion of the apparent front, drastically changing the 'answer' ($P_{\text{off-line}}$) given by our EC search. Again, this suggests that we not rely on a single multicriterion EC approach, but instead build up (piece together) a tradeoff surface by taking the best (i.e. Pareto-optimal subset) of all solutions discovered by several different methods, EC and non-EC alike (see e.g. Ritzel *et al* 1994).

In general, we are asking our multicriterion EC algorithms to help us decide on our multicriterion preferences as well as to search for the best solutions under those preferences. These two objectives are not separable. Knowing the actual tradeoffs (solutions) available, that is knowing the results of a multicriterion EC search, can help a DM make difficult multicriterion decisions, but making multicriterion decisions up front can help the search for the best solutions. For now it appears that, as in the early years of single-objective EC, practitioners of multicriterion EC are best off experimenting with a number of different methods on a particular problem, both in terms of hybridizing complementary algorithms (above), and in terms of independent checks against each other's performances. On difficult, real-world multicriterion search problems, the most successful approaches will probably be those that (i) incorporate domain- and problem-specific knowledge, including DM preferences, and (ii) use EC searches interactively to build up knowledge of the tradeoffs available and the tradeoffs desired.

F1.9.5 Alternative approaches

It is clear that EC is well suited to multicriterion problem solving because there are few, if any alternatives. Certainly the field of multiobjective decision analysis offers no competitive search technique, relying mostly on enumeration, or assuming only linear interactions among decision variables, to allow tractable, deterministic search. The traditional alternatives to EC-based optimization (e.g. simulated annealing (SA), stochastic hillclimbing, tabu search, and other robust, stochastic search algorithms) could be substituted for an EC algorithm if the multicriterion problem can be reduced to a single-criterion problem, via aggregation. For example, Cieniawski (1993) separately tries SA, a domain-tailored *branch and bound* heuristic (MICCP), and a GA, as single-objective stochastic optimizers in multiple independent runs using scalar aggregations (both a linear combination and the nonlinear Tscbycheff weighting function) to sample the Pareto front. (He finds that the GA finds approximately the same frontier as SA and MICCP.) However if one wishes to perform *multicriterion search*, to determine the entire *Pareto-optimal frontier* (the optimal tradeoff surface), as a number of recent studies have succeeded in doing, then EC algorithms, with their large populations and ability to search *partially ordered* spaces, seem uniquely suited to the task, and they bring to bear the known (if not quantified) benefits of EC search, including implicit and massive parallel processing and tolerance of noisy, uncertain objective functions.

G9.7.4

Acknowledgement

Most of the research for this contribution was performed by the author while in residence at the Illinois Genetic Algorithms Laboratory (IlliGAL) at the University of Illinois at Urbana-Champaign (UIUC). The author acknowledges support provided by NASA under contract number NGT-50873, the US Army under contract DASG60-90-C-0153, and the National Science Foundation under grant ECS-9022007, while at the IlliGAL at UIUC.

References

Baita F, Mason F, Poloni C and Ukovich W 1995 Genetic algorithm with redundancies for the vehicle scheduling problem *Evolutionary Algorithms for Management Applications* ed J Biethahn and V Nissen (Berlin: Springer) pp 341–53

Baker J E 1985 Adaptive selection methods for genetic algorithms *Proc. Int. Conf. on Genetic Algorithms and Their Applications (ICGA)* ed J J Grefenstette (Hillsdale, NJ: Erlbaum) pp 101–11

Bhanu B and Lee S 1994 *Genetic Learning for Adaptive Image Segmentation* (Boston, MA: Kluwer)

Belegundu A D, Murthy D V, Salagame R R and Constans E W 1994 Multi-objective optimization of laminated ceramic composites using genetic algorithms *Proc. 5th AIAA/NASA/USAF/ISSMO Symp. on Multidisciplinary Analysis and Optimization (Panama City, FL, 1994)* (AIAA) pp 1015–22

Chang C S, Wang W, Liew A C, Wen F S and Srinivasan D 1995 Genetic algorithm based bicriterion optimisation for traction substations in DC railway system *Proc. 2nd IEEE Int. Conf. on Evolutionary Computation (Perth, November–December 1995)* vol 1 (Piscataway, NJ: IEEE) pp 11–16

Cieniawski S E 1993 *An Investigation of the Ability of Genetic Algorithms to Generate the Tradeoff Curve of a Multi-objective Groundwater Monitoring Problem* Master's Thesis, University of Illinois at Urbana-Champaign

Cieniawski S E, Eheart J W and Ranjithan S 1995 Using genetic algorithms to solve multiobjective groundwater monitoring problem *Water Resource Res.* **31** 399–409

De Jong K A 1975 *An Analysis of the Behavior of a Class of Genetic Adaptive Systems* Doctoral Dissertation, University of Michigan; *Dissertation Abstracts Int.* **36** 5140B; University Microfilms 76-9381

de Neufville R 1990 *Applied Systems Analysis: Engineering Planning and Technology Management* (New York: McGraw-Hill)

Eheart J W, Cieniawski S E and Ranjithan S 1993 *Genetic Algorithm-Based Design Groundwater Monitoring System* WRC Research Report 218, Water Resources Center, University of Illinois at Urbana-Champaign

Fonseca C M and Fleming P J 1993a Genetic algorithms for multiobjective optimization: formulation, discussion and generalization *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 416–23

——1993b *Multiobjective Genetic Algorithms* (London: IEE) pp 6/1–6/5

——1994 Multiobjective optimal controller design with genetic algorithms *Proc. IEE Control '94 Int. Conf. (Warwick, 1994)* vol 1 *(IEE Conf. Publication 389)* (London: IEE) pp 745–9

——1995a An overview of evolutionary algorithms in multiobjective optimization *Evolutionary Comput.* **3** 1–16

——1995b *Multiobjective Optimization and Multiple Constraint Handling with Evolutionary Algorithms I: a Unified Formulation* Research Report 564, Department of Automatic Control and Systems Engineering, University of Sheffield

——1995c *Multiobjective Optimization and Multiple Constraint Handling with Evolutionary Algorithms II: Application Example* Research Report 565, Department of Automatic Control and Systems Engineering, University of Sheffield

——1995d Multiobjective genetic algorithms made easy: selection, sharing and mating restriction *Proc. 1st IEE/IEEE Int. Conf. on Genetic Algorithms in Engineering Systems: Innovations of Applications (GALESIA '95) (IEE Conf. Publication 414)* (London: IEE) pp 44–52

Fourman M P 1985 Compaction of symbolic layout using genetic algorithms *Proc. 1st Int. Conf. on Genetic Algorithms (Pittsburgh, PA, July 1985)* ed J J Grefenstette (Hillsdale, NJ: Erlbaum) pp 141–53

Gero J S and Louis S J 1995 Improving Pareto optimal designs using genetic algorithms *Microcomput. Civil Eng.* **10** 239–47

Goldberg D E 1989 *Genetic Algorithms in Search, Optimization, and Machine Learning* (Reading, MA: Addison-Wesley)

Goldberg D E and Richardson J J 1987 Genetic algorithms with sharing for multimodal function optimization *Proc. 2nd Int. Conf. on Genetic Algorithms (Cambridge, MA, 1987)* ed J J Grefenstette (Hillsdale, NJ: Erlbaum) pp 41–9

Greenwood G, Hu X and D'Ambrosio J G 1997 Fitness functions for multiobjective optimization problems: combining preferences with Pareto rankings *Foundations of Genetic Algorithms, IV* ed R K Belew and M D Vose (San Francisco, CA: Morgan Kaufmann) at press

Hilliard M R, Liepins G E, Palmer M and Rangarajen G 1989 The computer as a partner in algorithmic design: automated discovery of parameters for a multi-objective scheduling heuristic *Impacts of Recent Computer Advances on Operations Research* ed R Sharda, B L Golden, E Wasil, O Balci and W Stewart (New York: North-Holland) pp 321–31

Horn J and Nafpliotis N 1993 *Multiobjective Optimization using the Niched Pareto Genetic Algorithm* IlliGAL Report 93005, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign

Horn J, Nafpliotis N and Goldberg D E 1994 A niched Pareto genetic algorithm for multiobjective optimization *Proc. 1st IEEE Conf. on Evolutionary Computation (Orlando, FL, June 1994)* (Piscataway, NJ: IEEE) pp 82–7

Hwang C-L, Lai Y-J and Liu T-Y 1993 A new approach for multiple objective decision making *Comput. Operations Res.* **20** 889–99

Ishibuchi H and Murata T 1996 Multi-objective genetic local search algorithm *Proc. 3rd IEEE Int. Conf. on Evolutionary Computation* (Piscataway, NJ: IEEE) pp 119–24

Keeney R and Raiffa H 1976 *Decisions with Multiple Objectives* (New York: Wiley)

Kita H, Yabumoto Y, Mori N and Nishikawa Y 1996 Multiobjective optimization by means of the thermodynamical genetic algorithm *Proc. 4th Int. Conf. on Parallel Problem Solving from Nature (Berlin, 1996) (Lecture Notes in Computer Science 1141)* ed H-M Voigt, W Ebeling, I Rechenberg and H-P Schwefel (Berlin: Springer) submitted

Krause M and Nissen V 1995 On using penalty functions and multicriteria optimization techniques in facility layout *Evolutionary Algorithms for Management Applications* ed J Biethahn and V Nissen (Berlin: Springer) pp 153–66

Kundu S and Kawata S 1996 AI in control system design using a new paradigm for design representation *Proc. 4th Int. Conf. on Artificial Intelligence in Design* (Boston, MA: Kluwer)

Kundu S, Kawata S and Watanabe A 1996 A multicriteria approach to control system design with genetic algorithms *Int. Federation of Automatic Control (IFAC) '96—13th World Congr. (San Francisco, CA, 1996)* (Kidlington: Pergamon) at press

Kursawe F 1990 *Evolutionsstrategien für die Vektoroptimierung* Diplomarbeit, Universität Dortmund (in German)

——1991 A variant of evolution strategies for vector optimization *Parallel Problem Solving from Nature (PPSN) (Lecture Notes in Computer Science 496)* (Berlin: Springer) pp 193–7

Langdon W B 1995 Evolving data structures using genetic programming *Proc. 6th Int. Conf. on Genetic Algorithms (Pittsburgh, PA, July 1995)* ed L J Eshelman (San Mateo, CA: Morgan Kaufmann) pp 295–302

——1996 Using data structures within genetic programming *Genetic Programming 1996* ed J R Koza, D E Goldberg, D B Fogel and R L Riolo (Cambridge, MA: MIT Press) pp 141–9

Li Y, Gen M and Ida K 1996 Evolutionary computation for multicriteria solid transportation problem with fuzzy numbers *Proc. 3rd IEEE Int. Conf. on Evolutionary Computation* (Piscataway, NJ: IEEE) pp 596–601

Liepins G E, Hilliard M R, Richardson J and Palmer M 1990 Genetic algorithms applications to set covering and traveling salesmen problems *Operations Research and Artificial Intelligence: the Integration of Problem-Solving Strategies* ed D E Brown and C C White (Boston, MA: Kluwer) pp 29–57

Louis S and Rawlins G J E 1993 Pareto optimality, GA-easiness, and deception *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 118–23

Michielssen E and Weile D S 1995 Electromagnetic system design using genetic algorithms *Genetic Algorithms in Engineering and Computer Science* ed G Winter, J Périaux, M Galán and P Cuesto (Chichester: Wiley) ch 18, pp 345–69

Mori N, Yoshida J, Tamaki H, Kita H and Nishikawa Y 1995 A thermodynamical selection rule for the genetic algorithm *Proc. 2nd IEEE Int. Conf. on Evolutionary Computation (Perth, November–December 1995)* vol 1 (Piscataway, NJ: IEEE) pp 188–92

Murata T and Ishibuchi H 1995 MOGA: multi-objective genetic algorithms *Proc. 2nd IEEE Int. Conf. on Evolutionary Computation (Perth, November–December 1995)* vol 1 (Piscataway, NJ: IEEE) pp 289–94

Osyczka A 1984 *Multicriterion Optimization in Engineering (with FORTRAN Programs)* (Chichester: Ellis Horwood)

Osyczka A and Kundu S 1995 A new method to solve generalized multicriteria optimization problems using the simple genetic algorithm *Structural Optimization* **10** 94–9

——1996 A modified distance method for multicriteria optimization, using genetic algorithms *Comput. Industrial Eng. J.* **30** at press

Pareto V 1896 *Cours d'Économie Politique, 1* (Lausanne: Rouge)

Poloni C 1995 Hybrid GA for multi objective aerodynamic shape optimisation *Genetic Algorithms in Engineering and Computer Science* ed G Winter, J Périaux, M Galán and P Cuesto (Chichester: Wiley) pp 397–415

Richardson J T, Palmer M R, Liepins G and Hilliard M 1989 Some guidelines for genetic algorithms with penalty functions *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, June 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 191–7

Ritzel B J, Eheart W and Ranjithan S 1994 Using genetic algorithms to solve a multiple objective groundwater pollution containment problem *Water Resources Res.* **30** 1589–603

Savic D A and Walters G A 1995 Genetic operators and constraint handling for pipe network optimization *Evolutionary Computing (AISB Workshop, Sheffield, 1995) (Springer Lecture Notes in Computer Science (LNCS) 993)* ed T C Fogarty (Berlin: Springer) pp 154–65

Schaffer J D 1984 *Some Experiments in Machine Learning using Vector Evaluated Genetic Algorithms* Doctoral Dissertation, Department of Electrical Engineering, Vanderbilt University, unpublished

——1985 Multiple objective optimization with vector evaluated genetic algorithms *Proc. 1st Int. Conf. on Genetic Algorithms (Pittsburgh, PA, July 1985)* ed J J Grefenstette (Hillsdale, NJ: Erlbaum) pp 93–100

Schaffer J D and Grefenstette J J 1985 Multi-objective learning via genetic algorithms *Proc. 9th Int. Joint Conf. on Artificial Intelligence* vol 1, pp 593–5

Shaw K J and Fleming P J 1996a Initial study of multi-objective genetic algorithms for scheduling the production of chilled ready meals *2nd Int. Mendel Conf. on Genetic Algorithms (Mendel '96) (Brno, 1996)*; Research Report 623, Department of Automatic Control and Systems Engineering, University of Sheffield

Simpson A R, Dandy G C and Murphy L J 1994 Genetic algorithms compared to other techniques for pipe optimization *J. Water Resources Planning Management* **120** 423–43

Srinivas N 1994 *Multiobjective Optimization using Nondominated Sorting in Genetic Algorithms* Master's Thesis, Indian Institute of Technology

Srinivas N and Deb K 1995 Multiobjective optimization using nondominated sorting in genetic algorithms *Evolutionary Comput.* **2** 221–48

Stanley T J and Mudge T 1995 A parallel genetic algorithm for multiobjective microprocessor design *Proc. 6th Int. Conf. on Genetic Algorithms (Pittsburgh, PA, July 1995)* ed L J Eshelman (San Mateo, CA: Morgan Kaufmann) pp 597–604

Steuer R E 1986 *Multiple Criteria Optimization: Theory, Computation, and Application* (New York: Wiley)

Surrey P D, Radcliffe N J and Boyd I D 1995 A multi-objective approach to constrained optimisation of gas supply networks: the COMOGA method *Evolutionary Computing (AISB Workshop, Sheffield, 1995) (Springer Lecture Notes in Computer Science (LNCS) 993)* ed T C Fogarty (Berlin: Springer) pp 166–80

Takada Y, Yamamura M and Kobayashi S 1996 An approach to portfolio selection problems using multi-objective genetic algorithms *Proc. 23rd Symp. on Intelligent Systems* pp 103–8 (in Japanese)

Tamaki H, Kita H and Kobayashi S 1996 Multi-objective optimization by genetic algorithms: a review *Proc. 3rd IEEE Int. Conf. on Evolutionary Computation* (Piscataway, NJ: IEEE) pp 517–22

Tamaki H, Mori M, Araki M, Mishima Y and Ogai H 1995 Multicriteria optimization by genetic algorithms: a case of scheduling in hot rolling process *Proc. 3rd Conf. of the Association of Asian–Pacific Operational Research Societies (APORS '94)* ed M Fushimi and K Tone (Singapore: World Scientific) pp 374–81

Thierens D 1995 *Analysis and Design of Genetic Algorithms* Doctoral Dissertation, Catholic University of Leuven

Thierens D and Goldberg D E 1994 Elitist recombination: an integrated selection recombination GA *Proc. 1st IEEE Conf. on Evolutionary Computation (Orlando, FL, June 1994)* (Piscataway, NJ: IEEE) pp 508–12

Tsoi E, Wong K P and Fung C C 1995 Hybrid GA/SA algorithms for evaluating trade-off between economic cost and environmental impact in generation dispatch *Proc. 1st IEEE Int. Conf. on Evolutionary Computation (Orlando, FL, June 1994)* vol 1 (Piscataway, NJ: IEEE) pp 132–7

Vemuri V R and Cedeño W 1995 A new genetic algorithm for multi-objective optimization in water resource management *Proc. 1st IEEE Int. Conf. on Evolutionary Computation (Orlando, FL, June 1994)* vol 1 (Piscataway, NJ: IEEE) pp 495–500

Wallace D R, Jakiela M J and Flowers W C 1994 Design search under probabilistic specifications using genetic algorithms *Computer-Aided Design* **28** 405–20

Wienke D, Lucasius C and Kateman G 1992 Multicriteria target vector optimization of analytical procedures using a genetic algorithm (Part I. Theory, numerical simulations and application to atomic emission spectroscopy) *Anal. Chim. Acta* **265** 211–25

Yoshida K, Yamamura M and Kobayashi S 1996 Generating a set of Pareto optimal decision trees by GA *Proc. 4th Int. Conf. on Soft Computing (1996)* at press

F1.10 Simulated evolution

Paul Devine and Raymond C Paton

Abstract

Here we consider simulated evolution (SE) in the context of natural and biological systems rather than in purely computational or engineering terms, these having been dealt with elsewhere. SE often uses evolutionary computation (EC) methodologies but equally may use others, related or nonrelated. We consider the application of these techniques to various problems within this domain of natural evolution and compare and contrast their respective strengths and weaknesses.

F1.10.1 Introduction

Though work in simulated evolution (SE) has expanded considerably over the past two decades the simulation of evolution on a digital computer has been subject to investigation for almost as long as the computer has existed. Early work included the exploration of the theory of automata and their replication and possible evolution by Von Neumman and Burks (1966). In the fifties and sixties people such as Fraser (1957), Fogel *et al* (1966), and Reed *et al* (1967) addressed the topic. For example, the work of Reed *et al* (1967) with the FREDERIC system utilized numeric patterns and the now common *genetic operations* of crossover and mutation. Holland (1975) provided much of the seminal work on adaptation and his many contributions included the *genetic algorithm* (GA). Since the mid-1970s there has been an enormous growth and diversification of work in SE. This activity has been fueled in no small part by the growing interest in complex systems as a multidisciplinary area and the availability of significant computing resources to non-computer specialists to explore simulations within their own fields. Real, biological evolution most commonly takes place over very long timescales; consequently many facets are generally not well suited to a conventional process of direct experimentation. The possibility of simulating evolution on computers has been recognized as a potential tool for studying some of the processes and principles of evolution in ways that provide novel perspectives. Over and above this, SE may also provide the means for effective ecological and ethological modeling and simulation.

B1.2

Collins (1994) defines artificial evolution as being specifically concerned with the use of GAs. We prefer the term simulated evolution to cover a much broader area of research, encompassing other methods both within and outside evolutionary computation. In saying this, the concepts of variation, population, and selection are as central to SE theory as they are to evolutionary theory.

SE, in both EA (evolutionary algorithm) and non-EA guises, has been used to investigate phenomena ranging from the origin of life through to ethology. The work involved, by its very nature, is multidisciplinary and pragmatic, and does not invite simple categorization. In order to appreciate the breadth of work that has been carried out in the SE field we shall discuss it in terms of two categories of approach: EA and non-EA.

The strict EA approach as envisaged here uses, for example, GAs (Holland 1975), *genetic programming* (GP) (Koza 1993), *evolutionary programming* (EP) (Fogel 1993), and *evolution strategies* (ESs) (Rechenberg 1973, Schwefel 1981) as the basis of evolutionary change. All of these methods may be characterized as conforming to a broad selectionist paradigm (Darden and Cain 1989) and reflect biological evolution on varying levels and to varying degrees. Leaving aside the independent origins of these algorithms their salient practical differences may be summarized as follows.

B1.5.1

B1.4, B1.3

GAs, EP, and ESs are the most heavily biological of the four and may be viewed as reflecting different aspects of evolution. GAs parallel the genetic level and commonly use a *string encoding* which is subject to the genetic operators *crossover* and *mutation*. EP and ESs more closely resemble the phenotype level of selection: individuals are selected to persist into the next round or generation. As EP is not constrained by the representational issues involved in GAs it tends to be far more flexible in this respect. These may be used in conjunction with other, non-EA, techniques to form a subset we term the hybrid EA approach. Common applications of GAs are to artificial neural nets (ANNs) (Collins and Jefferson 1991) and cellular automata (CAs) (Sims 1992). There is also the closely related approach that employs mutatable code which runs on a virtual machine. Though sharing GP features it is, nevertheless, distinctly different in that it uses mutation rather than crossover as the source of variation in the population. C1.2 C3.3.1, C3.2

GP employs populations of *tree-structured* programs, these being composed of primitives and terminals. Crossover is the genetic operator applied and this allows the searching of the space of potential solution programs. C1.6

The non-EA approach simply encompasses those not defined by the above. CAs and autocatalytic sets are good examples of methodologies that have yielded novel and interesting results (Langton 1991, Bagley and Farmer 1991). Kampis (1994) proposes a string processing language (SPL) for the modeling of biological and evolutionary processes: this is based on the need for a shift in perspective on lifelike computing and the representation of novelty.

F1.10.2 The problem domain

The problem domain is that of biological evolution. This encompasses the whole spectrum from the origin of life to the evolution of complex behaviors. The fundamental precept of the process of evolution is the action of selection on a population of individuals which contains variation, there should also be a source of new variation. The level at which selection occurs is not crucial to SE, though it could certainly be used to investigate hypotheses on the subject. Some candidates for the application of SE techniques are:

- biology
  - origin of life; self-organization, emergence, and self-replication (Langton 1991)
  - speciation (Yaeger 1993)
  - origin of sex (Collins and Jefferson 1992)
  - life history effects (Linton *et al* 1991)
  - relationship between phenotype and genotype (Schuster 1991)
  - information processing and signalling in cells (Bray and Lay 1994, Chiva and Tarroux 1994)
- ethology
  - behavioral ecology (Sumida *et al* 1990)
  - evolutionary game theory (Kurka 1992)
  - cooperative behavior and eusociality (Koza 1994)
- ecology
  - implementation of computational ecologies as a modeling medium (Forrest and Jones 1994)
  - relationship between learning and evolution (Ackley and Littman 1991)
  - simulation of real ecological systems.

F1.10.3 The evolutionary algorithm route to simulated evolution

As EAs were inspired by evolution it is only natural that they should be suited to the implementation of artificial evolutionary systems. One of the main EAs for this sort of application is the GA (Holland 1975). This is directly derived from biology and its representations and operators have attracted considerable research interest and effort.

The molecular level is the lowest at which evolution has been simulated with GAs with a degree of biological verisimilitude. Schuster (1991) developed an artificial RNA based around binary strings, selection being centered on an evaluation of phenotype (which depended on genotype in environment) rather than directly on the genotype.

Two particularly good examples of the application of GAs to the more general questions posed by evolutionary theory are the work of Sumida *et al* (1990) and Gibson (1989). The former use a GA to investigate a number of ecological issues including deme structure, migration rates, and the division of time between foraging and singing time in birds. The latter employs an intentionally simplified GA to

look at at the effects of mutation rates and breeding strategies, amongst other things, on the performance of GAs on relatively simple problems. He shows that biological behavior can emerge in even very simple systems.

At a higher evolutionary level, GAs have been used extensively in conjunction with ANNs and classifier systems as the vehicles of adaptive change. Polyworld (Yaeger 1993) uses ANNs specified by GAs as the basis for virtual creatures in an totally abstract world; these creatures exhibit behavioral divergence and speciation over time. Collins and Jefferson (1991) have used a similar arrangement as the adaptive basis of a simulation of an ant colony in order to study the circumstances of emergent social behavior; this contrasts with the previous example as it reflects a real system as opposed to the *de novo* approach of polyworld and others. Further ant-colony-related work has been carried out with GP (Koza 1994). Here evolution of the type of emergent behavior observed in social and eusocial animals was modeled. Though giving little insight into the real evolutionary processes due to the method used, this work demonstrates how the evolution of simple behaviors may lead to much more complex overall behaviors.

Evolutionary game theory has also proven a fertile area for the application of GAs. For example, a number of facets of natural evolution have emerged from noisy games of iterated prisoners' dilemma where the strategies have been genetically encoded and manipulated (Lindgren 1991).

Standard computer code is not suitable for the direct application of mutation: it is far too brittle and mutations usually result in inoperable programs as opposed to new, executable programs. In order to circumvent this problem a number of systems have been written which consist of a virtual machine on which specially developed machine code is run. This code is extremely robust in order to support mutation. Examples of this approach are Tierra (Ray 1991), C-Zoo (Skipper 1992), and Avida (Adami and Brown 1995). These differ in detail and the dimensionality of the virtual environment but are all essentially bottom-up approaches to evolution.

Tierra, as the best known example, has an evolvable instruction set that runs on a virtual MIMD (multiple-instruction-stream, multiple-data-stream) machine. The ecological simulations performed have thrown up such 'biological' behavior as the emergence of host–parasite and even superparasite relationships; Lotka–Voltera type cycling between parasite and host is also observable (Ray 1991).

F1.10.4 The non-evolutionary-algorithm route to simulated evolution

Automata in general, and *cellular automata* in particular, are perhaps the longest-standing members of the SE menagerie. Von Neumann speculated on evolution in self-replicating automata nearly half a century ago. They have been used extensively in biological modeling but their static nature does not lend itself easily to the simulation of evolutionary processes. This has not prevented their application, especially at a fundamental conceptual level where they have been used to look at the emergence of lifelike behavior. Langton has published research suggestive of the emergence of lifelike organization at phase transitions (Langton 1991). G1.6

Holland (1976) proposed α-universes as a class of abstract systems with sufficient complexity to support the emergence of lifelike behavior; this is in essence a CA type system. Holland's own work was largely theoretical in nature, but it has been followed up experimentally (McMullin 1992). The most striking aspect of this work from the SE point of view is the possible emergence of self-replicating complexes with genetic-like descriptions of themselves, though not from the exact framework postulated by Holland.

Taking the somewhat broader definition of evolution given by Spencer, given by Bagley *et al* (1991) as part of their motivation, we come to another non-EA approach, autocatalytic sets. This views evolution as the process of long-term organizational change in nature. Though there is no unique representation here the most heavily used has been a connectionist, graph representation. This work is essentially an investigation of self-organization and the consequent 'chemical evolution' has shown the development of autocatalytic and metabolic behavior. This suggests possible mechanisms by which life may have occurred.

F1.10.5 Conclusion

We have dealt with SE largely in terms of EA and non-EA categories but do not pretend that these are unequivocally drawn boundaries. Our main concern is with the biologically inspired. EAs are fundamentally selectionist in nature but possess differing degrees of biological influence. Much of the

work detailed here is illustrative rather than exhaustive. It would seem that the various methodologies are often suited to differing levels and areas of enquiry, though some certainly pervade many different areas. However, generally speaking we would make the following comments.

GAs are widely used, due in part to their biological motivation and in part to the fact that they are fairly well understood. They lend themselves well to use in conjunction with other approaches where appropriate representations are used. ES and EP predate GAs; they are most usually applied to optimization problems and are more representationally flexible than GAs. GP is perhaps less biological in its conception and is less suited to biological applications.

Aside from its applications in engineering and combinatorial problems SE has considerable potential for the exploration of the sort of biological phenomena mentioned in this article, and probably more besides. Those techniques that have been derived from EA work pervade the field and have produced many interesting results. They are often best used in conjunction with other techniques, ANNs and CAs being particularly good cases in point. Perhaps the biological inspiration of EAs makes them well suited to this type of work.

However, we would add some important caveats when considering the efficacy of computer simulation and modeling. Observed phenomena in simulations may well seem to correlate well with biological observation but this similarity is not sufficient to assume a common cause. This is especially a problem at the most highly abstracted levels of CAs and autocatalytic sets. There is always the danger that we are learning about our computer models rather than biological reality. Additionally, EAs are often treated as optimization techniques and their application to evolution is in part predicated on the assumption that evolution is itself a form of optimization. Though this viewpoint is widespread it is important to remember that this is not the only viewpoint and is certainly not universally accepted.

References

Ackley D and Littman M 1991 Interactions between learning and evolution *Artificial Life II* ed J D Farmer, C G Langton, S Rasmussen and C Taylor (Reading, MA: Addison-Wesley) pp 488–509

Adami C and Brown C T 1995 Evolutionary learning in the 2D artificial system 'Avida' *Artificial Life IV: Proc. 4th Int. Workshop on the Synthesis and Simulation of Living Systems* ed C G Langton *et al* pp 377–82

Bagley R J and Farmer J D 1991 Spontaneous emergence of a metabolism *Artificial Life II* ed J D Farmer, C G Langton, S Rasmussen and C Taylor (Reading, MA: Addison-Wesley) pp 93–135

Bagley R J, Farmer J D and Fontana W 1991 Evolution of a metabolism *Artificial Life II* ed J D Farmer, C G Langton, S Rasmussen and C Taylor (Reading, MA: Addison-Wesley) pp 141–58

Bray D and Lay S 1994 Computer simulated evolution of a network of cell-signaling molecules *Biophys. J.* **66** 972–7

Chiva E and Tarroux P 1994 Studying genotype/phenotype interactions: a model of the evolution of the cell regulation network *Proc. Int. Conf. on Evolutionary Computation (PPSNIII) (Lecture Notes in Computer Science 866)* ed Y Davidor, H-P Schwefel and R Männer (Berlin: Springer) pp 26–35

Collins R J 1994 Artificial evolution and the paradox of sex *Computing with Biological Metaphors* ed R Paton (London: Chapman and Hall) pp 244–63

Collins R J and Jefferson D R 1991 Antfarm: towards simulated evolution *Artificial Life II* ed J D Farmer, C G Langton, S Rasmussen and C Taylor (Reading, MA: Addison-Wesley) pp 579–601

——1992 The evolution of sexual selection and female choice *Towards a Practice of Autonomous Systems* ed F J Varela and P Bourgine (Cambridge, MA: MIT Press) pp 327–36

Darden L and Cain J A 1989 Selection type systems *Phil. Sci.* **56** 106–29

Davidor Y 1994 Free the spirit of evolutionary computing: the ecological genetic algorithm paradigm *Computing with Biological Metaphors* ed R Paton (London: Chapman and Hall) pp 311–22

Findlay S and Rowe G 1990 Computer experiments on the evolution of sex: the haploid case *J. Theor. Biol.* **146** 379–93

Fogel D B 1993 On the philosophical foundations of evolutionary algorithms and genetic algorithms *Proc. 2nd Annu. Conf. on Evolutionary Programming (San Diego, CA, 1993)* ed D B Fogel and W Atmar (La Jolla, CA: Evolutionary Programming Society) pp 23–9

Fogel L J, Owens A J and Walsh M J 1966 *Artificial Intelligence through Simulated Evolution* (New York: Wiley)

Forrest S and Jones T (1994) Modeling complex adaptive systems with Echo *Complex Systems: Mechanism of Adaptation* ed R J Stonier and X H Yu (Amsterdam: IOS) pp 3–21

Fraser A S 1957 Simulation of genetic systems by automatic digital computers *Aust. J. Biol. Sci.* **10** 484–91

Gibson J M 1989 Simulated evolution and artificial selection *Biosystems* **23** 219–28

Holland J H 1975 *Adaptation in Natural and Artificial Systems* (Cambridge, MA: MIT Press)

——1976 Studies of the spontaneous emergence of self-replicating systems using cellular automata and formal grammars *Automata, Languages, Development* ed A Lindenmayer and G Rosenberg (New York: North-Holland)

Kampis G 1994 Life-like computing beyond the machine metaphor *Computing with Biological Metaphors* ed R Paton (London: Chapman and Hall) pp 392–413

Koza J R 1993 *Genetic Programming: on the Programming of Computers by means of Natural Selection* (Cambridge, MA: MIT Press)

——1994 Evolution of emergent cooperative behavior using genetic programming *Computing with Biological Metaphors* ed R Paton (London: Chapman and Hall) pp 280–99

Kurka P 1992 Natural selection in a population of automata *Towards a Practice of Autonomous Systems* ed F J Varela and P Bourgine (Cambridge, MA: MIT Press) pp 375–82

Langton C G 1991 Life at the edge of chaos *Artificial Life II* ed J D Farmer, C G Langton, S Rasmussen and C Taylor (Reading, MA: Addison-Wesley) pp 41–89

Lindgren 1991 Evolutionary phenomena in simple dynamics *Artificial Life II* ed J D Farmer, C G Langton, S Rasmussen and C Taylor (Reading, MA: Addison-Wesley) pp 295–312

Linton L, Sibly R M and Calow P 1991 Testing life-cycle theory by computer simulation. Introduction of genetical structure *Comput. Biol. Med.* **21** 345–55

McMullin B 1992 The Holland α-universes revisited *Towards a Practice of Autonomous Systems* ed F J Varela and P Bourgine (Cambridge, MA: MIT Press) pp 317–26

Ray T S 1991 An approach to the synthesis of life *Artificial Life II* ed J D Farmer, C G Langton, S Rasmussen and C Taylor (Reading, MA: Addison-Wesley) pp 371–408

Rechenberg I 1973 *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien biologischen Evolution* (Frommann-Holzboog)

Reed J, Toombs R and Barricelli N 1967 Simulation of biological evolution and machine learning *J. Theor. Biol.* **17** 319–42

Schwefel H 1981 *Numerical Optimisation of Computer Models* (Chichester: Wiley)

Schuster P 1991 Complex optimisation in an artificial RNA world *Artificial Life II* ed J D Farmer, C G Langton, S Rasmussen and C Taylor (Reading, MA: Addison-Wesley) pp 277–90

Sims K 1992 Interactive evolution of dynamical systems *Towards a Practice of Autonomous Systems* ed F J Varela and P Bourgine (Cambridge, MA: MIT Press) pp 171–6

Skipper J 1992 The computer zoo—evolution in a box *Towards a Practice of Autonomous Systems* ed F J Varela and P Bourgine (Cambridge, MA: MIT Press) pp 355–64

Sumida B H, Houston A I, McNamara J M and Hamilton W D 1990 Genetic algorithms and evolution *J. Theor. Biol.* **147** 59–84

Von Neumann J and Burks A W 1966 *Theory of Self-reproducing Automata* (University of Illinois Press)

Yaeger L 1993 Computational genetics, physiology, metabolism, neural systems, learning, vision and behavior *Artificial Life III* ed C Langton *et al* (Reading, MA: Addison-Wesley)

G1.1 Designing a reduced-complexity algorithm for quaternion multiplication

David Beasley

Abstract

This case study describes how a reduced-complexity algorithm for quaternion multiplication can be devised using a genetic algorithm (GA). This design task is highly epistatic, and difficult for a conventional GA to tackle. Consequently, rather than having a simple representation, simple operators, and a simple fitness function, but a highly epistatic search space, a new technique is used to spread the task's complexity more evenly. Using this new technique, known as *expansive coding*, the representation, operators, and fitness function become more complicated, but the search space becomes less epistatic, and therefore easier for a GA to tackle. In the design of a multiplier for quaternion numbers, consistently good results are obtained using this technique.

G1.1.1 Introduction

G1.1.1.1 Project overview

The work described in this article was carried out as part of a doctoral research programme which investigated the suitability of *genetic algorithms* (GAs) for a task relevant in digital signal processing: algorithm simplification (Beasley 1995). A traditional type of generational replacement GA was used, known as *GAT*. GAT is based on the simple GA of Goldberg (1989), and was implemented in Pop-11 especially for this research. B1.2

The task for the GA was to design a simple algorithm for multiplying two quaternions (see below). The GA has to minimize the number of real-number multiplications required in the algorithm. To be a valid solution, an algorithm has to perform quaternion multiplication *exactly*, not merely approximately.

G1.1.1.2 Quaternion multiplication

Quaternions (Hamilton 1899, Brand 1947, Martin 1983) are numbers with *four* components; they may be written as $q \equiv a + \mathrm{i}b + \mathrm{j}c + \mathrm{k}d$, where a, b, c and d are real numbers, and i, j and k are the three *quaternion operators*. Each of these is analogous to the complex number operator, i.

If two quaternions, $q_1 \equiv (a + \mathrm{i}b + \mathrm{j}c + \mathrm{k}d)$ and $q_2 \equiv (p + \mathrm{i}q + \mathrm{j}r + \mathrm{k}s)$ are multiplied to give $(w + \mathrm{i}x + \mathrm{j}y + \mathrm{k}z)$, then the components to be computed are

$$w = (ap - bq - cr - ds) \tag{G1.1.1}$$

$$x = (aq + bp + cs - dr) \tag{G1.1.2}$$

$$y = (ar - bs + cp + dq) \tag{G1.1.3}$$

$$z = (as + br - cq + dp). \tag{G1.1.4}$$

Hence there is a trivial algorithm for multiplying two quaternion numbers which requires 16 real-number multiplications. The task of the GA is to devise an algorithm which uses fewer multiplications. If this can be done, implementations can be improved.

G1.1.1.3 The importance of representation

In any GA, the choice of task *representation* is crucial to a successful outcome. Difficult tasks can be made easier to solve if the representation is designed so that gene interaction (*epistasis*) is low. Finding a suitable representation can be especially difficult for combinatorial optimization tasks, especially where only a few points in the search space represent feasible solutions, and the rest have zero (or undefined) fitness. Such 'all-or-nothing' tasks are very difficult to solve, and cannot generally be tackled by a GA using a direct representation. C1

Several ideas have been suggested to overcome this difficulty. What is required is to allocate a fitness value to the infeasible solutions, in such a way that they will lead the GA towards feasible ones. Methods have been suggested (Cramer 1985, De Jong and Spears 1989, Richardson *et al* 1989), but they do not always give good results (Beasley 1995, chapter 4).

A new approach for solving combinatorial tasks using GAs has instead been developed, known as *expansive coding*. In this article the technique of expansive coding is described in the context of applying it to simplifying an algorithm for quaternion multiplication.

G1.1.2 Expansive coding

The central idea of expansive coding is to split a large, highly epistatic task into a number of subtasks, so that even though high epistasis may remain *within* each subtask, epistasis *between* subtasks is lower. Regions of high epistasis are thus localized. In any task, in the limit, as the size of epistatic regions decreases towards a single bit, a task becomes trivially easy to solve. Hence, by localizing regions of high epistasis, the task can be expected to be easier to solve. Subtask validity can be ensured by appropriate coding and operators. This leaves the GA with the greatly simplified task of arranging relatively weakly interacting subsolutions to find the overall solution of highest fitness.

The steps involved in designing the coding and the GA can be described as follows.

- *Splitting.* The task is split into subtasks, so that any combination of valid subsolutions gives a valid overall solution. Subtask representations are concatenated together to form a chromosome. (Obviously, this may not always be feasible.)
- *Local constraints.* These must be placed on each subtask, to ensure that the subsolutions represented are always valid. They can be enforced either by using a careful coding scheme which makes it impossible to represent invalid subsolutions, or by using, instead of conventional crossover and mutation, task specific operators which always maintain the validity of a subtask.
- *Local fitness.* In some tasks any valid subsolution may be just as fit as any other. In other tasks, however, it may be possible to assign a partial fitness value to each subsolution in isolation. For example, in a three-dimensional bin packing task, we may prefer to place heavier objects at the bottom of bins.
- *Merging algorithm.* The major fitness calculation comes from attempting to merge the subsolutions back together again into a single, global solution. Methods for doing this will be task specific. The more merging that is successfully carried out, the fewer distinct subsolutions will remain, and the higher the fitness. The total fitness is computed from some combination (e.g. the weighted sum) of the merge fitness and the local fitnesses.

Even in the absence of local fitness values, subsolutions which are easy to merge with other subsolutions will tend to increase in the population. Also, sensible *ordering* of the subtasks can help promote the creation of building blocks. In these ways, the population converges towards a set of coherent subsolutions.

G1.1.2.1 Reproduction operators

Operators must be designed to maintain the validity of subsolutions. To prevent *crossover* from disrupting the validity of any subsolution, it will often be necessary to restrict crossing sites to lie between subtasks. Similarly, *mutation* operators will normally work on one symbol (i.e. a whole subtask) at a time, rather than on one bit at a time. C3.3.1 C3.2

G1.1.3 Tackling the quaternion multiplier task

Quaternion multiplication may be viewed as the task of mapping four input variables, (a, b, c, d), to four output variables, (w, x, y, z), via sixteen multipliers, with (p, q, r, s) as parameters. For example, from equation (G1.1.1), the mapping $a \rightarrow w$ is achieved by multiplying by the value p. This can be represented as a linear signal flow graph, as shown in figure G1.1.1, where triangles represent multiplication, and circles represent addition.

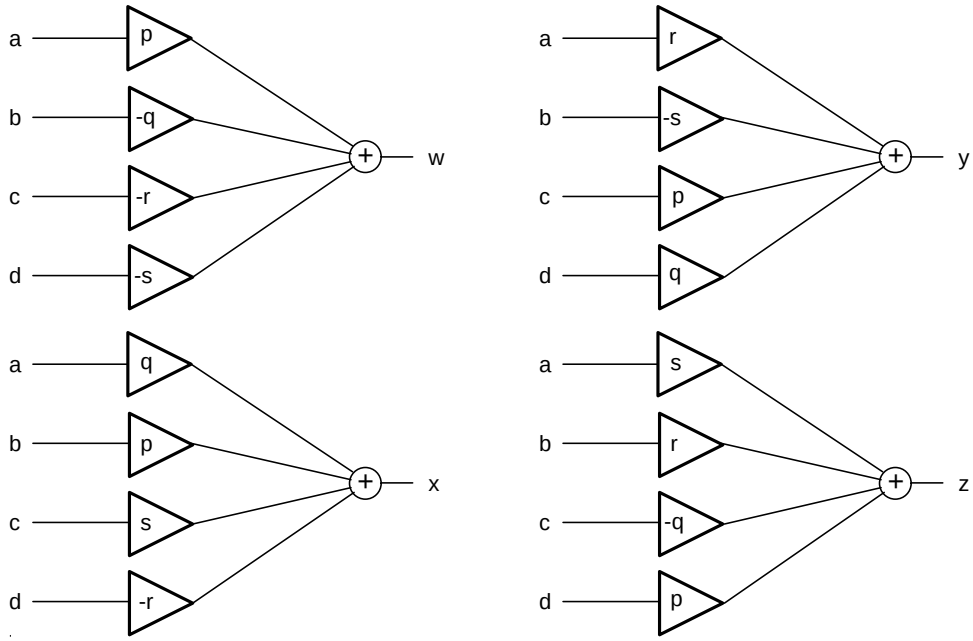


Figure G1.1.1. Simple quaternion multiplication.

In figure G1.1.1, each multiplier is used exactly once. However, because of redundancy, it is possible to *reuse* multipliers, so that inputs and/or outputs can *share* multipliers, reducing the total number needed. A generic circuit arrangement for this sharing scheme is shown in figure G1.1.2. Here there are n multipliers, which multiply by factors $g_1, g_2, g_3, \ldots, g_n$. Each of these factors, or *gains*, g_i, can be specified by four coefficients, $h_p(g_i), h_q(g_i), h_r(g_i)$, and $h_s(g_i)$, such that $g_i = h_p(g_i)p + h_q(g_i)q + h_r(g_i)r + h_s(g_i)s$. The input of each multiplier is connected via a set of *input links* to each of the input nodes. Each input link represents a *potential* connection between an input node, and an adder/subtracter unit at the input to a multiplier. Each input link therefore represents a connection with a gain in $\{-1, 0, 1\}$. Similarly, the output of each multiplier is connected to each output node via an *output link*, with a gain in $\{-1, 0, 1\}$.

With this arrangement, it is possible for several inputs to share a common multiplier, and also for the output of one multiplier to be shared among several outputs. By a suitable choice of input and output link gains, and multiplier gains, it is possible to represent a broad class of input–output transfer functions, a subset of which will correctly perform quaternion multiplication. The lower the value of n, the higher the fitness of the circuit. The task to solve is, what is the minimum value of n, and what gain values are required to achieve this?

G1.1.4 Applying a genetic algorithm

Clearly this task is difficult for a GA. If a direct representation scheme were to be used, in which values for input link gains, output link gains, and multiplier gains were all simply coded into the chromosome, there would be a very high degree of epistasis. Changing any multiplier gain value would, potentially, alter *all* of the input–output transfer functions. Hence it would be impossible to improve a reasonably good chromosome by making small changes to it. No building blocks could ever form, since the fitness of any subgroup of genes is highly dependent on the values of most of the other genes. To overcome these problems the expansive coding technique was used, as detailed below.

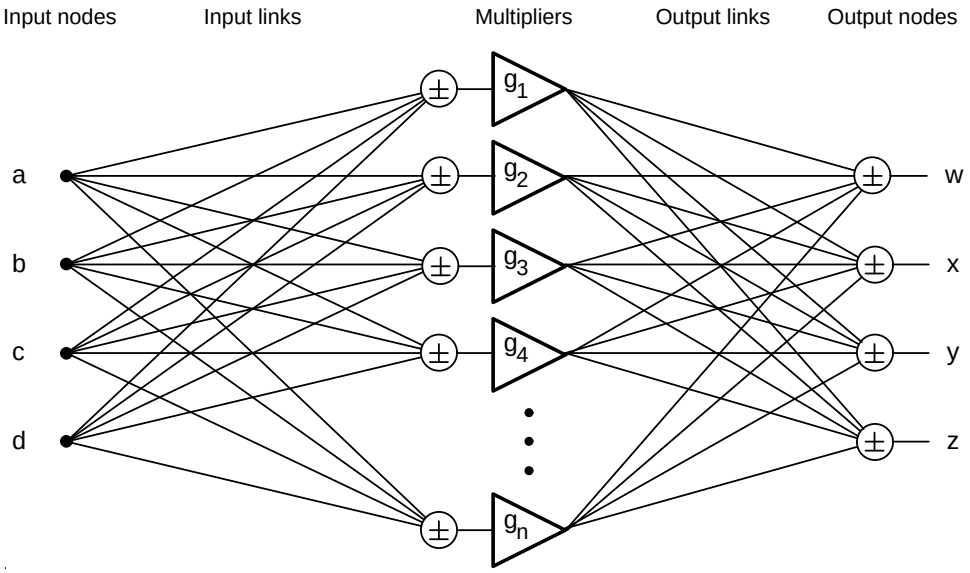


Figure G1.1.2. Generic circuit.

G1.1.4.1 Splitting

The task is split into sixteen subtasks, one for each input–output transfer function. Each subtask has S multipliers of its own with which to fulfil the correct transfer function. As far as the representation is concerned, these multipliers are *not* shared with any other subtasks. For the $a \rightarrow w$ transfer function, shown in figure G1.1.3, the multipliers have gains $g_{aw1}, g_{aw2}, g_{aw3}, \ldots, g_{awS}$, and the mapping is therefore

$$w_a = a \sum_{i=1}^{S} g_{awi}. \tag{G1.1.5}$$

The total output is given by $w = w_a + w_b + w_c + w_d$. The other 15 mappings are treated in the same way, which means the chromosome must hold information for $16S$ multipliers.

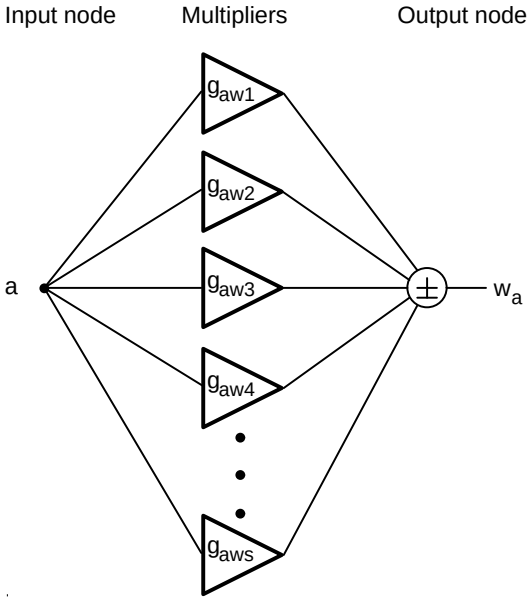


Figure G1.1.3. Circuit for one subtask.

Each multiplier in figure G1.1.3 can be represented by four gain coefficients, h_p, h_q, h_r, and h_s (figure G1.1.5(c)). For simplicity, it is assumed that each of these will be in $\{-1, 0, 1\}$. Therefore, two bits are needed for each coefficient, or eight bits per multiplier (figure G1.1.5(d)). Input and output link gains do not need to be represented explicitly. An input or output link gain of zero can is equivalent to a multiplier gain of zero. Similarly, link gains of ± 1 can be absorbed by changing the overall sign of the multiplier gain. The required link gain values can be deduced during the merging phase (see below).

G1.1.4.2 Local constraints

Having split the task into sixteen subtasks, we now consider what local constraints should be applied to each of these. For a subtask mapping input $u \in \{a, b, c, d\}$ to output $v \in \{w, x, y, z\}$, the transfer function is

$$v_u = p\sum_{i=1}^{S} h_p(g_{uvi}) + q\sum_{i=1}^{S} h_q(g_{uvi}) + r\sum_{i=1}^{S} h_r(g_{uvi}) + s\sum_{i=1}^{S} h_s(g_{uvi}) \tag{G1.1.6}$$

or, putting $H(u, v, p) \equiv \sum_{i=1}^{S} h_p(g_{uvi})$ and so on, this becomes

$$v_u = pH(u, v, p) + qH(u, v, q) + rH(u, v, r) + sH(u, v, s). \tag{G1.1.7}$$

Equations (G1.1.1)–(G1.1.4) give the total gains required in each of the 16 cases. For example, for the $a \to w$ mapping, we require $H(a, w, p) = 1$, $H(a, w, q) = 0$, $H(a, w, r) = 0$, $H(a, w, s) = 0$. This means that within each subtask, the sums of the gain coefficients, h_p, h_q, h_r, and h_s, must be maintained at specific values. To achieve this, chromosomes in the initial population are set up with valid sums, and the operators used are designed to maintain this validity (see below).

G1.1.4.3 Local fitness

In each subtask, any set of gain values conforming to equations (G1.1.1)–(G1.1.4) will be valid and, in isolation, each is equally good. Thus, local fitness values are not used.

G1.1.4.4 The merging algorithm

The merging algorithm must bring together multipliers which have equal gains and compatible input–output connections. For example, suppose two multipliers each have a gain of $(p + q)$, (that is, $h_p = 1$, $h_q = 1$, $h_r = 0$, and $h_s = 0$). If one has an input connection to a, the other an input connection to b, and both have output connections to w, then they may be merged. This would give a single multiplier with gain $(p + q)$, its output connected to w and its input the sum (or difference) of a and b. This is shown in figure G1.1.4.

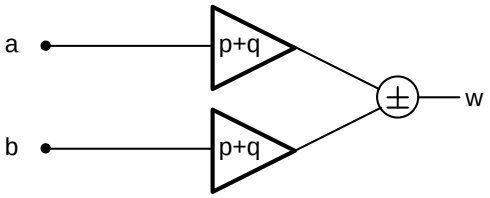
In general, there will be several different ways of merging a set of multipliers, so to find the optimum merging pattern an exhaustive search must be done. This is a slow process, but fortunately an *approximate* fitness evaluation method can be used (Goldberg 1989, pp 138, 206). A *greedy algorithm* finds an optimal merging pattern in most cases, so our GA uses this to determine an approximate fitness for each chromosome during a run. Only when the GA has converged and a solution has been found is the exhaustive search algorithm used to determine the exact fitness.

After merging, the number of distinct multipliers with non-zero gain is taken as the fitness value. The GA must minimize this value.

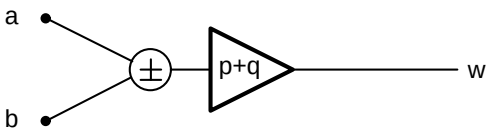
G1.1.4.5 Representation

Careful organization of the chromosome allows building blocks to form. A set of subsolutions is well adapted if many of their multipliers can be merged. If they are also close together on the chromosome, they can form a building block. Merging can only take place between multipliers which share common input or output connections. So, a chromosome organization is needed where subtasks are close together if they share common inputs, *or* if they share common outputs. This cannot be achieved with a conventional one-dimensional chromosome, but is easily arranged on a two-dimensional chromosome.

The most natural organization is therefore a 4×4 array of the subtasks (figure G1.1.5(a)) where each subtask is represented by S multipliers (figure G1.1.5(b)). Each *row* of the array contains subtasks relating



(a) Before Merging



(b) After Merging

Figure G1.1.4. Illustration of merging.

to the same output node. Conversely, each *column* contains subtasks relating to the same input node. Merging can therefore only take place between multipliers in the same row or column, since multipliers must share a common input or common output node. Consequently, it is possible for building blocks to form as coherent rows or columns evolve.

G1.1.4.6 Reproduction operators

Crossover. To avoid creating invalid chromosomes, crossover points are only allowed at subtask boundaries. Three different crossover operators are used. *Whole-row* and *whole-column* crossover are two-dimensional projections of normal *two-point crossover*. Two cut points are chosen, and complete rows (or columns) are swapped over between the parents to produce two offspring. In *square-patch* crossover, the chromosome is treated as a torus, and two row cut points and two column cut points are chosen, defining a square patch. This is then swapped from one parent to the other, giving two offspring. C3.3.1

Mutation. Two operators are used. Both first select a subtask to work on. A multiplier is chosen, and one or more of its gain coefficients is incremented or decremented. A record is kept of the alteration made. To regain the validity of the subsolution, another multiplier is chosen, and compensating decrements or increments are made to its gain coefficients. As pointed out above, the effect is to maintain the sums of the gain coefficients, h_p, h_q, h_r, and h_s, at specific values. Occasionally it proves to be impossible to make the required change to a selected multiplier (because the change would take a gain coefficient outside the allowed range -1 to $+1$), in which case another multiplier is chosen, and that changed instead.

The two mutation operators differ in how the alteration is made to the first multiplier chosen. The *swap component* mutation operator increments or decrements one of the multiplier's gain coefficients by one. The *zeroize gain* operator sets the multiplier gain to zero. The latter operator will tend to reduce the number of multipliers in use in the chromosome, and therefore create a pressure to simplify the algorithm.

G1.1.5 Method

Two methods of fitness scaling were tried: *linear*, with truncation to zero for individuals whose expected reproductive count would have been below zero (similar to *sigma truncation* (Goldberg 1989, p 124)); and *ranked*, with linearly allocated expected offspring values. In both cases, the number of reproductive opportunities for the most fit individual in each generation was 2.0. A crossover probability of 0.8 was used, and mutation probability of 0.064 per subtask (giving an average of 1.0 mutations per chromosome, for $S = 8$.) To produce a pair of offspring, one of the crossover operators was chosen and applied, then one of the mutation operators was chosen, and applied to each child. The probabilities of use of each operator were held fixed during each run. A run was terminated when the population average fitness,

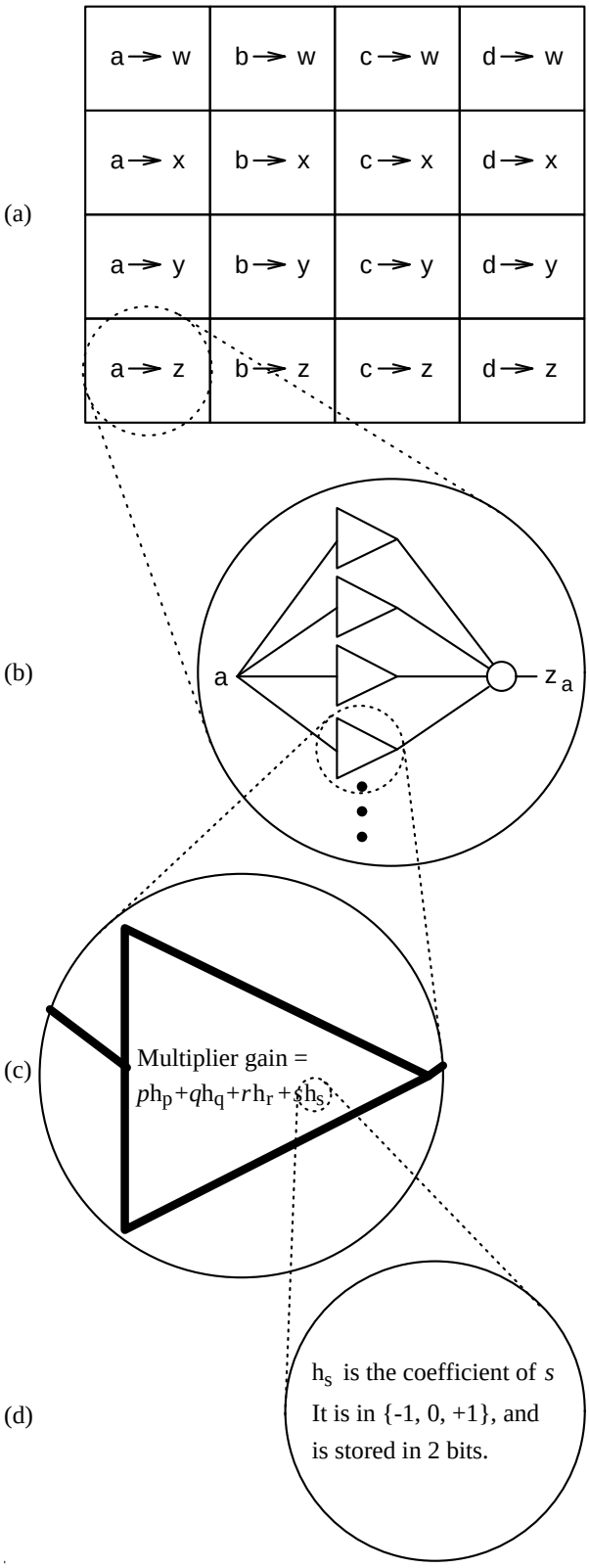


Figure G1.1.5. Chromosome organization: (*a*) a 4 × 4 subtask array; (*b*) a single subtask; (*c*) a single multiplier; (*d*) a single gain coefficient.

measured over a moving window of a fixed number of generations, stopped increasing. For crossover, an equal probability for each of the three operators was used. For mutation, the probabilities were weighted 10:3 in favour of swap component mutation, to encourage exploration.

G1.1.6 Results

Many trials were performed with $S = 8$. This gives a chromosome of 1024 bits, and a search space of approximately 10^{150} valid chromosomes. The best solutions found had only ten multipliers. A typical ten-multiplier solution is shown in figure G1.1.6. The results for different population sizes, averaged over 100 runs, are summarized in table G1.1.1.

The effectiveness of the expansive coding technique is clearly demonstrated. Every run came up with a solution better than the obvious 16-multiplier arrangement. With larger populations, very good ten-multiplier solutions were found regularly. The best population size out of those tried appears to be 200, and linear truncated fitness scaling performs better than ranking.

G1.1.7 Conclusions

At first sight, expansive coding seems counterintuitive, since it makes the search space much larger. The task is, however, made simpler, since the interaction (epistasis) between the elements which the GA has to manipulate (the subtasks) is reduced.

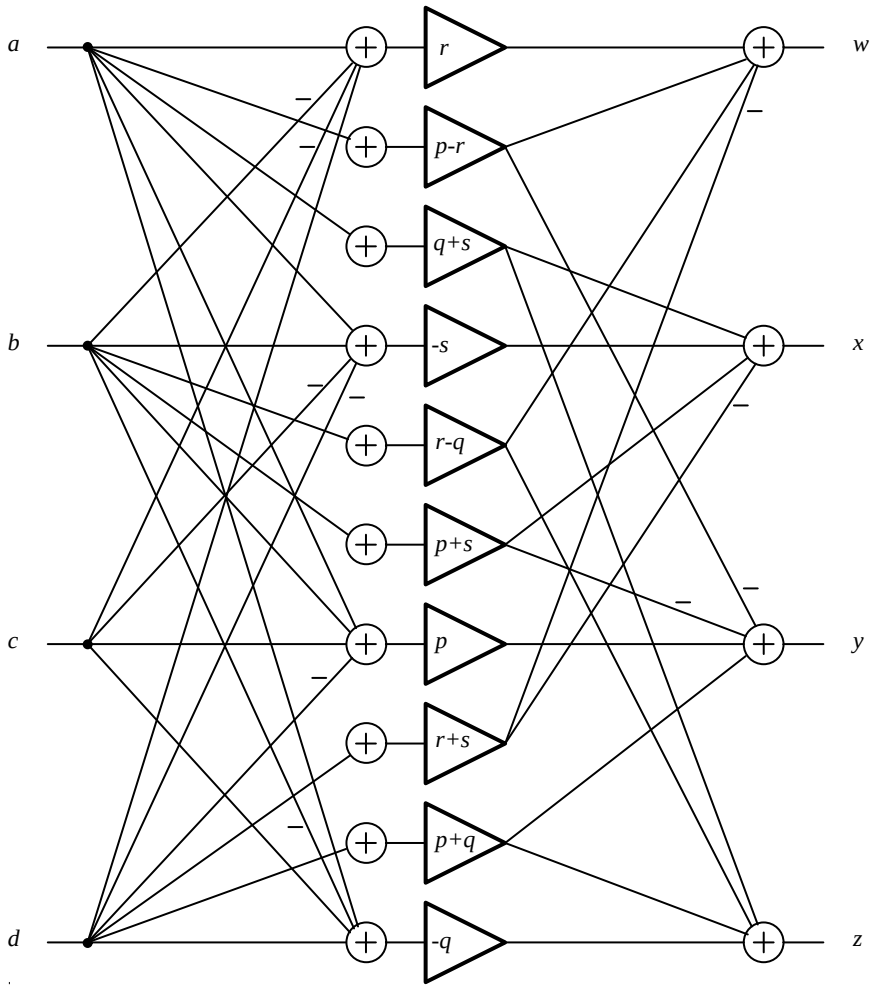


Figure G1.1.6. A ten-multiplier solution.

Table G1.1.1. Percentage of runs finding a ten-multiplier solution.

Scaling	Pop. size	Window size	Evals./run	Worst solution	% success
ranking	100	30	14 000	15 mults.	1
ranking	200	60	55 800	13 mults.	24
ranking	400	100	155 000	13 mults.	32
linear	100	30	16 100	15 mults.	2
linear	200	60	56 600	13 mults.	33
linear	400	100	147 700	13 mults.	58

With appropriate representation and operators, the inherent complexity of a task may be shifted, so that, although the fitness decoding function becomes more complicated, the GA finds the task easier. In theory, this allows any task to be made trivially easy to solve, from the point of view of the GA (Vose and Liepins 1991). GAs are good for tasks of intermediate epistasis (Davidor 1990). On highly epistatic tasks, therefore, a suitable representation and operator set must be found which sufficiently reduces the epistasis. The expansive coding technique is one such approach. Complexity, in terms of epistasis in the original task, is traded for complexity in terms of an increased chromosome size, a more complicated fitness function, and the need for task specific operators. One large dose of complexity has therefore been split into three smaller doses.

The application of this method to algorithm simplification shows that it can be highly effective. This task area does not require absolutely optimal solutions at great speed; if a technique can improve upon existing designs, then it is useful.

References

Beasley D 1995 *Expansive Coding: a Representation Method for Arithmetic Algorithm Optimisation Using Genetic Algorithms* PhD Thesis, University of Wales College of Cardiff

Beasley D, Bull D R and Martin R R 1993 Reducing epistasis in combinatorial problems by expansive coding *Proc. 5th Int. Conf. on Genetic Algorithms* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 400–7

Brand L 1947 *Vector and Tensor Analysis* (New York: Wiley)

Cramer N L 1985 A representation for the adaptive generation of simple sequential programs *Proc. 1st Int. Conf. on Genetic Algorithms* ed J J Grefenstette (Erlbaum) pp 183–7

Davidor Y 1990 Epistasis variance: suitability of a representation to genetic algorithms *Complex Systems* **4** 369–83

De Jong K and Spears W M 1989 Using genetic algorithms to solve NP-complete problems *Proc. 3rd Int. Conf. on Genetic Algorithms* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 124–32

Goldberg D E 1989 *Genetic Algorithms in Search, Optimization and Machine Learning* (Reading, MA: Addison-Wesley)

Hamilton W R 1899 *Elements of Quaternions* (Cambridge: Cambridge University Press)

Martin R R 1983 Rotation by quaternions *Math. Spectrum* **17** 42–8

Richardson J T, Palmer M R, Liepins G E and Hilliard M R 1989 Some guidelines for genetic algorithms with penalty functions *Proc. 3rd Int. Conf. on Genetic Algorithms* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 191–7

Vose M and Liepins G 1991 Schema disruption *Proc. 4th Int. Conf. on Genetic Algorithms* ed R K Belew and L B Booker (San Mateo, CA: Morgan Kaufmann) pp 237–42

G1.2 Exploiting constraints as background knowledge for evolutionary algorithms

Jan Paredis

Abstract

This case study describes the use of constraint programming within evolutionary algorithms (EAs) to solve *constrained optimization problems* (COPs). The approach presented here is called *genetic state-space search* (GSSS). It integrates two general search paradigms: genetic search and state-space search. GSSS uses domain knowledge in the form of constraints to limit the space to be searched by the EA. GSSS searches for 'promising search states' from which good solutions can easily be found. Moreover, GSSS allows the handling of constraints within the genetic search at a general, domain-independent level. First, a genetic representation of search states is introduced. Next, the operation of GSSS is explained. Finally, a job shop scheduling application of GSSS is described.

G1.2.1 Introduction

Constrained optimization problems (COPs) typically consist of a set of n variables x_i $(1 \leq i \leq n)$ which have an associated domain D_i of possible values. In addition, there is also a set C of constraints which describe relations between the values of the x_i (e.g. $x_1 \neq x_3$). Finally, an objective function f is given. An *optimal solution* consists of an assignment of values to the x_i such that (i) all constraints in C are satisfied, that is, the solution is *valid*, and (ii) the assignment yields an optimal value for the objective function f.

A typical *constraint program* proceeds as follows. First, a *constraint network* is generated. This involves the creation of the variables x_i, their domains D_i, and the constraints between the variables. Figure G1.2.1 depicts a constraint network. The nodes of this graph represent variables; links correspond to constraints. In this figure, there is a constraint between each pair of variables. After the construction of the constraint network, a constraint-based search algorithm repeats the following *selection–assignment–propagation* cycle (SAP cycle): select a variable whose domain contains more than one element, select a value from the domain of that variable, assign the chosen value to the chosen variable (i.e. the variable's domain becomes a singleton containing this value), and finally perform propagation. This propagation process executes all constraints defined on the variable whose domain is reduced. This might further reduce the domain of other variables.

The search algorithm described above—known as *forward checking*—can easily be described as a standard state-space search. A set of potential solutions can be associated with each search state. This set is simply the product of all domains, that is, $D_1 \times D_2 \times \ldots \times D_N$. At each choice point (assignment), the domain of a variable is reduced to a singleton, followed by constraint propagation. This reduces the set of solutions associated with a state because the domains become smaller. Whenever a domain becomes empty, no solution exists for the given choices, or, in other words, the current state is a dead end. In this case, the algorithm backtracks. When finally all domains are reduced to a singleton, a solution has been found.

Constraint programming has established itself as a suitable technique for solving combinatorial problems. The use of constraints often reduces the amount of backtracking considerably. This is because

dead ends can be detected at an early stage. Another advantage of constraint programming is that one can state the constraints of the problem domain in a natural way. A good introduction to constraint programming can be found in the book by Van Hentenrijck (1989).

G1.2.2 Design process

G1.2.2.1 Motivation for the approach

COPs typically exhibit a high degree of *epistasis*: the choices (assignments of values to variables) made during the search are closely coupled. In general, highly epistatic problems are characterized by the fact that no decomposition into independent subproblems is possible. In that case, it is difficult to combine subparts of two valid solutions into another valid solution. That is why the efficiency of *genetic algorithms* (GAs)—which typically search by combining features of different 'solutions'—decreases significantly for problems with higher degrees of epistasis. Problems with a high degree of epistasis are difficult for other types of EA—such as *evolution strategies* and *evolutionary programming*—as well. For maximally epistatic problems, a small change to a solution has a big impact on the fitness, or, in other words, the *fitness landscape* is uncorrelated. As we will see in this case study, constraints can be used at three stages present in all types of EA.

B1.2
B1.3, B1.4
B2.7

A number of methods for constraint handling have been proposed within the EA community. The first one, *genetic repair*, removes constraint violations in invalid solutions generated by the genetic operators, such as mutation and crossover. The second one uses *decoders* such that all possible representations give rise to a valid solution. A third one uses *penalty functions*. This approach defines the fitness function as the objective function one tries to optimize minus the penalty function representing the 'degree of invalidity' of a solution. All three approaches are, however, problem specific: for every COP, one has to determine a good decoder, a good genetic repair method, or a penalty function that balances between convergence towards suboptimal valid solutions (when the penalty function is too harsh) or towards invalid solutions (when too tolerant a penalty function is used).

C5.4
C5.3
C5.2

In contrast to the three approaches mentioned above, GSSS does not focus on individual solutions. Instead, it operates on states in the search space. Each state implicitly represents the (possibly empty) set of valid solutions that can be reached from it. This allows us to relate our approach to the standard state-space search paradigm. That is why GSSS is considerably less problem dependent than the three approaches mentioned above. As a matter of fact, GSSS acquires its generality from its embedding in the standard state-space search paradigm.

A subclass of COPs, those involving only numerical linear (in)equalities, can be elegantly solved with EAs. For these problems, the space of valid solutions is known to be convex. Michalewicz and Janikow (1991) used this property to define genetic operators which always generate valid solutions.

Here, we tackle the general class of COPs. The problem description of a COP typically contains constraints implicitly describing which portions of the search space contain valid solutions. We discuss how these constraints, which come with the problem specification in any case, can be used to improve the search efficiency of an EA by limiting the search space it has to explore. Hence, the constraints provide cheap domain specific knowledge which can be used to augment domain-independent search methods such as EAs.

G1.2.2.2 Representation description

The individuals on which GSSS operates are search states. Hence, the genetic representation used by GSSS should be able to represent all possible search states of the COP. Our representation heavily relies on the fact that a search state is uniquely determined by the choices (i.e. assignments) made to reach it from the initial state. In GSSS, the ith gene in the 'gene string' represents the assignment of x_i. Consider, for example, a search state in which x_2, x_3, and x_7 are assigned the values 1, 5, and 2, respectively. This state is represented by the string ?15???2?. We use the term *PIG representation* when referring to this *partially instantiated genotype* representation. This representation heavily draws from the work of Hinton and Nowlan (1987). They introduced this representation for the completely different purpose of studying how individual learning can guide evolution.

G1.2.2.3 Constraints

In GSSS, constraints guide the EA in three ways: during the creation of the initial population, during reproduction, and during fitness evaluation. At all these stages two representations of a search state are used side by side: the constraint network and its corresponding PIG representation. This PIG string contains the same assignments as the network. Moreover, propagation keeps the domains of the not yet assigned variables consistent with these assignments. Figure G1.2.1 depicts a constraint network corresponding to the PIG string ?15????2?.

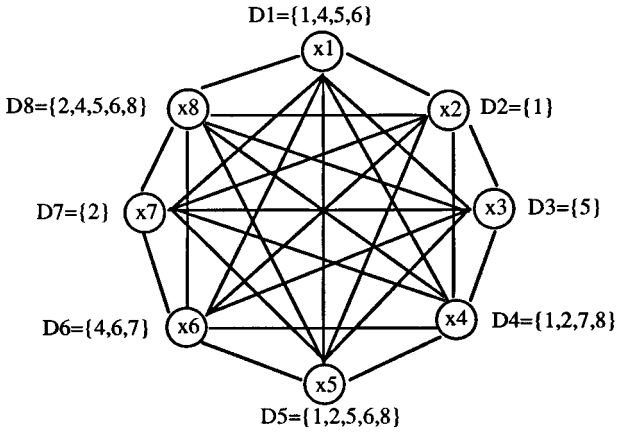


Figure G1.2.1. A constraint network corresponding to the PIG string ?15????2?.

The following three sections describe the three ways in which the constraints are used by the EA.

G1.2.2.4 Initial population

The creation of a search state belonging to the initial population starts with a PIG string containing only ?s and a constraint network with domains equal to those given in the problem specification. Next, a randomly chosen ? is filled in with a value randomly chosen from the domain of the corresponding variable in the constraint network. This assignment is followed by propagation in order to remove inconsistent values from the domains of other variables. This assignment–propagation process is repeated an *a priori* determined number of times. As a consequence, a member of the initial population typically consists of a PIG string with a fixed number of assignments.

G1.2.2.5 Operators

Constraints also play a vital role during crossover. First, a two-point crossover is applied on the PIG representation of the parents producing a PIG-represented child. Once again, the constraints can spot inconsistent assignments in this PIG representation and remove assignments (by changing them back into ?s) such that only a set of mutually consistent choices remains. This helps to avoid dead ends from which only invalid solutions can be reached. The same procedure can be used for mutation and for other crossover operators.

G1.2.2.6 Fitness function

We define the fitness of a search state as the value of the objective function for the best solution which can be reached from it through further assignments. Obviously, an exhaustive search through the set of potential solutions will often be far too expensive. Hence, GSSS explores only an *a priori* determined number of 'randomly chosen search paths'. The fitness is then the best solution found during these searches. Due to the stochastic nature of this search process, two different evaluations of the same individual can yield different fitness values. Notice that the fitness value is a lower bound of the best solution which can be reached from the search state. Along a search path, SAP cycles are executed starting from the constraint network corresponding to the PIG representation of the search state to be evaluated. The propagation

considerably limits the chances of ending up in a dead end. Hence, the chances of finding valid solutions are much higher than when no propagation is used. In this way, a better estimate of the promise of the state is obtained. The use of more intelligent (variable and value) selection heuristics—instead of the random ones proposed here—would obviously further improve the quality of the results.

G1.2.2.7 Results

The empirical results given by (Paredis 1993) allow us to understand the role and benefits of the use of constraints in the three components described above. This was done by comparing EAs which use the constraints in two of the three components (initial population, operators, and fitness calculation) with the algorithm which uses the constraints in all components. In addition to this, the 'constraint-free EA' which does not use constraints in any of the components was used as a baseline for comparison. All algorithms were tested on a well-known test problem: an optimization variant of the n-queens problem. Here, 30 queens have to be placed on a 30×30 chess board so that no two queens attack each other (i.e. they are not in the same row, column, or diagonal). Moreover, each position on the board has an associated randomly generated value. Now the goal is to find a valid solution such that the sum of the values of the occupied locations is maximized. The empirical results themselves will not be repeated here. Only the conclusions which can be drawn from them are discussed.

The use of the constraints during the fitness calculation is by far the most important. Without the guidance of the constraints, the fitness calculation performs purely random paths in the 'unpruned' search space. For this reason it is unlikely to find a valid—let alone an optimal—solution. In this case, the fitness calculation often considerably underestimates the fitness value of a search state. As a result many individuals from which good solutions can be reached may not have the chance to reproduce at all, or, in other words, their genes might be lost. This lack of focus during the genetic search not only causes a low average quality of solution, but is also responsible for the large variation between the best solution quality found in different runs.

Once constraints are used during the fitness calculation, a further improvement of the performance is obtained through the use of constraints during crossover. Now, crossover is much more likely to generate valid search states, that is, search states from which a good solution can be reached. The additional use of constraints during the creation of the initial population does not further improve the results. The initial individuals seem to act as a source of genetic diversity. It is not worthwhile to insist on initial validity because during reproduction the constraints steer towards 'valid' offspring.

G1.2.3 The job shop scheduling application

The general framework described in this paper grew out of earlier work on the use of constraint programming for *scheduling* (Paredis and Van Rij 1992). Now we discuss the use of GSSS for scheduling. F1.5, G9.4

G1.2.3.1 Job shop scheduling

The job shop scheduling problem can be formulated as follows: let $J_1, \ldots, J_n$ be the n jobs (orders) to be processed where each job has a *release date* and a *due date*; these dates indicate when the job is ready for processing and when it should be finished, respectively. Furthermore, there are m (unary) resources (e.g. machines) called $M_1, \ldots, M_m$. Unary resources can only work on one job at a time. Operation O_{ij} represents the processing of job J_i on resource M_j; p_{ij} is the duration of operation O_{ij}; and C is a set of constraints (see below). A job J_i is defined by the p_{ij} and the (linear) temporal sequence of the O_{ij}. A finished product, for example, must be painted before it is packed, and not the other way around. A valid schedule has start times assigned to the O_{ij} in such way that the constraints in C are satisfied. In our experiments, the goal is to find a schedule with minimal *makespan*, which is the overall length in time of the schedule.

G1.2.3.2 A constrained optimization problem representation for job shop scheduling

The representation of the job shop scheduling task as a COP is simple. Each variable x_i represents an operation. The domain of a variable represents the possible start times of the associated operation. This domain is initialized to the closed interval delimited by the earliest and latest possible start time of the

operation. These are computed from the release date and the due date of the job as well as from the durations of the operations belonging to the same job. The earliest possible start time of operation O_{ij}, for example, is the sum of the release date of job J_i and the durations of the operations of job J_i preceding O_{ij}. Likewise, the latest possible start time of operation O_{ij} is the due date of job J_i minus the sum of the duration of O_{ij} and of the operations of job J_i succeeding O_{ij}.

G1.2.3.3 Scheduling constraints

The most important type of constraint used here is the precedence constraint. In the initial problem statement, precedence constraints represent the job flow, that is, the precedence relations between each pair of operations belonging to the same job. Once a start time is chosen for an operation then the precedence constraint reduces the domain of a succeeding (preceding) operation such that it can start (finish), at the earliest (latest), when the preceding (succeeding) operation finishes (starts). Paredis (1992) gives a more thorough description of the constraints involved in job shop scheduling.

G1.2.3.4 Brief description of the algorithm and its results

The algorithm allows us to take into account the volatile environment in which scheduling takes place: orders may be canceled, machine breakdowns may occur, and the like. In such a volatile environment one should be able to reactively revise schedules in response to unexpected events. Instead of putting a large effort into finding one optimal schedule, we aim at finding search states from which a number of different good schedules can be reached. Whenever one cannot stick to a given good schedule, then one can—to some extent at least—search locally around these states for another feasible schedule. In order to achieve this, the fitness calculation takes into account not only the best solution found during the random searches but also the number of different solutions and the average quality of the solutions which can be reached from a search state. By changing the relative importance of these features the search process explores other regions of the search space trading off the density and variation of solutions, the quality of the best solution, and the average quality of the solutions. Due to space restriction the interested reader is referred to the article by Paredis (1992) for empirical results.

G1.2.4 Conclusion

GSSS combines the advantages of both EAs and constraint programming. EAs have proven to be good search algorithms for large, moderately epistatic, problems. Hence, we use genetic search to explore the large search spaces of COPs. A more thorough exploration of the search around a search state is made through constraint-based search. Domain knowledge, in the form of constraints, allows the EA to deal with higher degrees of epistasis.

Another advantage of GSSS is that constraint propagation is applied to states which are already somewhat constrained (i.e. a relatively large number of search choices are already made in the individuals). At these states, constraint propagation is particularly effective because it causes substantial reductions of the domains. Also note that this methodology can be used with any type of EA. Our experiments used a GENITOR-like algorithm (Whitley 1989).

The scheduling application shows that one can search for different types of search state. One is not restricted to search for states from which good solutions can be reached.

Paredis (1994) describes another approach using constraints within EAs. It uses a population of solutions which coevolves with a population of constraints. The interaction between the two populations is modeled after predator–prey interactions in nature. This approach uses somewhat less domain knowledge than GSSS because it only has to be able to test whether the constraints are satisfied by a solution. It does not need the knowledge to keep the domains consistent with the assignments.

References

Hinton G and Nowlan S J 1987 How learning can guide evolution *Complex Syst.* **1** 495–502

Michalewicz Z and Janikow C Z 1991 Handling constraints in genetic algorithms *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, July 1991)* ed R K Belew and L B Booker (San Mateo, CA: Morgan Kaufmann) pp 151–7

Paredis J 1992 Exploiting constraints as background knowledge for genetic algorithms: a case-study for scheduling *Proc. 2nd Conf. on Parallel Problem Solving from Nature (PPSN 92, Brussels, September 1992)* ed R Männer and B Manderick (Amsterdam: Elsevier) pp 229–38

——1993 Genetic state-space search for constrained optimization problems *Proc. 13th Int. Joint Conf. on Artificial Intelligence (IJCAI 93, Chambéry, August 1993)* ed R Bajcsy (San Mateo, CA: Morgan Kaufmann) pp 967–72

——1994 Coevolutionary constraint satisfaction *Parallel Problem solving from Nature—PPSN III (Proc. 3rd Conf. on Parallel Problem Solving from Nature, Jerusalem, October 1994)* ed Y Davidor, H-P Schwefel and R Männer (Berlin: Springer) pp 46–55

Paredis J and Van Rij T 1992 Simulation and constraint programming as support methodologies for job shop scheduling *J. Decision Syst.* **1** 59–77

Van Hentenrijck P 1989 *Constraint Satisfaction in Logic Programming* (Cambridge, MA: MIT Press)

Whitley D 1989 The Genitor algorithm and selection pressure: why rank-based allocation of reproductive trails is best *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, June 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 116–21

Further reading

1. Paredis J 1993 Genetic state-space search for constrained optimization problems *Proc. 13th Int. Joint Conf. on Artificial Intelligence (IJCAI 93, Chambéry, August 1993)* ed R Bajcsy (San Mateo, CA: Morgan Kaufmann) pp 967–72

   Integrates evolutionary algorithms and constraint programming at a problem-independent level.

2. Van Hentenrijck P 1989 *Constraint Satisfaction in Logic Programming* (Cambridge, MA: MIT Press)

   Provides a good introduction to constraint programming including various applications.

G1.3 Representing trees in genetic algorithms

Charles C Palmer and Aaron Kershenbaum

Abstract

There are many challenging problems in network design, for example, that may be modeled using trees. Trees are simple graphs of N nodes with $(N-1)$ edges interconnecting all of them. The traveling salesman problem and network topology optimization are examples of problems whose solutions take the form of trees. Tree problems are hard to solve optimally since as N grows the number of possible trees grows exponentially $\left(N^{N-2}\right)$, making enumerative searches impractical. The use of a genetic algorithm (GA) for these problems is attractive since GAs can search these large nonconvex spaces well. As with any GA application, the issue of representation of a solution, a tree in our case, on a chromosome is of primary importance. After comparing several commonly used representations and their usefulness in GAs, we describe a new representation and show it to be superior to the others in almost all respects. In particular, we show it can represent all possible trees, does not produce invalid trees after reproduction, and has good locality, all desirable features for the effective use of a GA. Through comparisons to traditional heuristic techniques and simulated annealing, we show that this new representation produces reliably good, if not optimal, solutions even when the problem definition changes.

G1.3.1 Introduction

In this case study, we consider the problem of representing trees in *genetic algorithms*. A tree is an undirected graph which contains no closed cycles. There are many optimization problems which can be phrased in terms of finding the optimal tree within a given graph or network. Sometimes any tree is permitted, as in the case of a minimum-spanning tree or shortest-path tree (Gondran and Minoux 1984). In other cases, the tree might be constrained. For example, the degrees of some or all of the nodes in the tree might be limited. While some problems using trees are easy, for many others (e.g. the Steiner tree problem, capacitated minimum-spanning tree problem, and *traveling salesman problem* (Gondran and Minoux 1984, Kershenbaum 1992, Gibbons 1985)) there is no practical means of obtaining a guaranteed optimal solution in a reasonable amount of time for large problems. In such cases, a genetic algorithm (GA) may be the best means available to obtain reliably good solutions to many variations (across different objective functions and constraints) of the problem.

B1.2

G9.5

In particular, we were interested in finding solutions to the optimal-communications spanning tree problem (OCSTP) (Hu 1974). The goal of this problem is to find a tree of minimum total cost satisfying a given requirement set. Thus we seek the tree, T, which minimizes

$$X(T) = \sum_{i,j} A_{ij}(T)\, C_{ij} \tag{G1.3.1}$$

where $A_{ij}(T)$ is the capacity required in T for the link between node i and j and C_{ij} is the link cost per unit capacity.

This problem is interesting for a number of reasons (Palmer 1994a). We choose to examine this problem here since it is a member of an intractable class of combinatorial optimization problems which

are not only NP-complete but also exhibit little 'locality' in their solution space. Most classical approaches to optimization problems rely on the fact that good solutions are 'close' to one another, and that one can move from one good solution to another by making small changes to the solution. These sorts of small change include adding a link, deleting a link, or exchanging a small number of links for one another. In the best cases, the solution space and the objective function are convex and we are guaranteed a globally optimal solution even when using a local search technique.

In this problem, however, this incremental approach does not work. Adding a link to a tree makes a cycle. Deleting a link from a tree disconnects the network. Exchanging one link for another changes the flows (and hence the costs) on many links and may also introduce a cycle. It is therefore necessary to use techniques such as GAs which are able to move between radically different solutions.

G1.3.2 Criteria for selecting tree representations in genetic algorithms

As described by Palmer (1994a, b), for a GA to function most effectively, the representation, or encoding, of trees must possess certain properties:

(i) It should be capable of representing all possible trees.
(ii) It should be unbiased in the sense that all trees are equally represented; that is, all trees should be represented by the same number of encodings. This property allows us to effectively select an unbiased starting population for the GA and gives the GA a fair chance of reaching all parts of the solution space.
(iii) It should be capable of representing only trees. To the extent that nontrees can be represented, it becomes more difficult to generate a random initial population for the GA. Worse yet, it becomes possible for crossover and mutation to produce nontrees from valid parent trees.
(iv) It should be easy to go back and forth between the encoded representation of the tree and the tree's representation in a more conventional form suitable for evaluating the fitness function and constraints.
(v) It should possess locality in the sense that small changes in the representation make small changes in the tree. This allows the GA to function more effectively by having the encoding truly represent the tree.

Ideally the representation of a tree for a GA should have all of these properties. Unfortunately, most representations trade some of these desirable traits for others. In the following sections, we will discuss a number of tree representations and evaluate them on the basis of how well they meet these criteria.

G1.3.3 Comparison of existing tree representations

We are given a set of N nodes which are labeled with the numbers $1, 2, \ldots, N$. In any specific problem, we are also given characteristics of the edges between the nodes such as their lengths. We denote the (undirected) edge between nodes i and j by (i, j). We assume here that all edges are possible candidates for inclusion in the tree; that is, that the underlying graph from which the tree is formed is a complete graph.

It has been shown (Moon 1967) that the number of possible trees in a complete graph on N nodes is $N^{(N-2)}$. Since each such tree can correspond to N possible rooted trees, with any node designated as the root, there are thus $N^{(N-1)}$ possible rooted trees. We can thus assess how efficient any representation is by comparing the number of graphs that can be represented by it to the possible number of trees. If the former is much larger than the latter, many nontrees can also be represented and this is, as we mentioned above, a problem.

G1.3.3.1 Characteristic vector

If we associate an index k with each link (i, j), we can represent a tree T as a vector $\boldsymbol{E} = e_k$, $k = 1, 2, \ldots K$, where K is the number of edges in the underlying graph and e_k is one if edge k is part of T and zero otherwise.

In a complete graph, $K = N(N-1)/2$. There are thus $2^{(N(N-1)/2)}$ possible values for $\boldsymbol{E}$ and, unfortunately, most of these are not trees. Indeed, since all trees have exactly $N-1$ edges, if $\boldsymbol{E}$ contains other than $N-1$ ones it is not a tree. Even if $\boldsymbol{E}$ contains exactly N ones, the probability of a random $\boldsymbol{E}$ being the representation of a tree is infinitesimally small as N increases (Palmer 1994b).

Thus, if we were to generate random vectors in order to provide a starting population for a GA, it is quite likely that none of them would be trees. Furthermore, when we mate any two trees in the course of a GA, it is very likely that none of the offspring would be trees.

It is an $O(N^2)$ effort to go back and forth between this encoding and a tree. This is not very good, since as we will see there are other methods where only $O(N)$ effort is required.

On the positive side, all trees can be represented by such vectors, all are represented equally (once), and the representation does possess a natural locality; changing a bit in the vector adds or deletes a single edge. On the whole, however, this is a poor representation for a GA because of the extremely low probability of obtaining a tree.

G1.3.3.2 Predecessors

An alternative representation is to designate a root, r, for the tree and then record the predecessor of each node in the tree rooted at r. Thus $\text{Pred}[i] = j$ if j is the first node in the path from i to r in T. Thus, every rooted tree T is represented by a unique N-'digit' number, where the 'digits' are numbers between 1 and N. We see, therefore, that this encoding is unbiased and covers the space of solutions.

There are N^N such numbers. Since there are $N^{(N-1)}$ rooted trees, a random number of this type represents a tree with probability $1/N$. This is a great improvement over the characteristic vector, but still allows for many nontrees being generated both in the initial population and during the breeding which takes place during the course of the GA.

Given a matrix mapping node pairs into edges, which can be set up at the start of the GA and requires $O(N^2)$ space, we can transform back and forth between this representation and a list of edges in $O(N)$.

G1.3.3.3 Prüfer numbers

A third possible encoding is the Prüfer number (Moon 1967) associated with a tree, defined as follows. Let T be a tree on N nodes. The Prüfer number, $P(T)$, is an $(N-2)$-'digit' number, where once again the digits are numbers between 1 and N . Assuming N is at least three, the algorithm to convert a Prüfer number into the unique tree it represents is as follows:

(i) Let $P(T)$ be the original Prüfer number and let all nodes not part of $P(T)$ be designated as eligible for consideration.

(ii) If no digits remain in $P(T)$, there are exactly two nodes, i and j, still eligible for consideration. (This can be seen by observing that as we remove a digit from $P(T)$ in step (iii) below, we remove exactly one node from consideration and there are $N-2$ digits in the original $P(T)$). Add (i, j) to T and stop.

(iii) Let i be the lowest-numbered eligible node. Let j be the leftmost digit of $P(T)$. Add the edge (i, j) to T. Remove the leftmost digit from $P(T)$. Designate i as no longer eligible. If j does not occur anywhere in what remains of $P(T)$, designate j as eligible.

(iv) Return to step (ii).

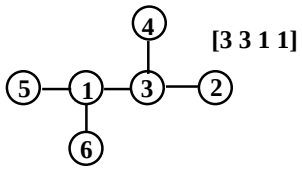


Figure G1.3.1. A tree and its Prüfer number.

Figure G1.3.1 shows the tree on six nodes corresponding to $P(T) = 3311$. A similar algorithm can be applied to obtain a Prüfer number for a given tree (Palmer 1994a).

There are $N^{(N-2)}$ Prüfer numbers for a graph with N nodes. This is exactly the number of trees possible in such a graph. There is, in fact, a one to one correspondence between trees and Prüfer numbers, the transformation being unique in both directions.

Thus, Prüfer numbers are unbiased (each tree is represented once), they cover the entire space of trees, and they do not represent anything other than trees. The transformations back and forth between edges and Prüfer numbers can be carried out in $O(N \log N)$ with the aid of a heap.

The disadvantage of this representation is that is has relatively little locality. While any offspring formed by taking parts of two Prüfer numbers will indeed be a tree, it need not resemble the parent trees at all. Indeed, changing even one digit of a Prüfer number can change the tree dramatically. Consider, for example, the six node trees formed from the Prüfer numbers 3241 and 3242 which have only two of their five edges in common.

We thus see that while each of these representations is acceptable by some of the criteria listed above, none is entirely adequate and it would be desirable to find a more suitable representation for trees within genetic algorithms. We present such a representation in the following section.

G1.3.4 A new representation for trees

Experience with the OCSTP has shown that for a given problem (nodes, requirements, and costs), certain nodes should be interior nodes and others should be leaf nodes. With this in mind, we designed a new encoding that allowed the GA to search for nodes with these tendencies while looking for solutions to the OCSTP. In this encoding, the chromosome holds a *bias* value for each node. For example, in a four-node problem the chromosome would contain four node biases $[b_1 b_2 b_3 b_4]$. These values are multiplied by a parameter P, and by the maximum link cost, $C_{\max}$, and are added to the cost of all links that have the node as an endpoint. The cost matrix is then biased by these values:

$$C'_{ij} = C_{ij} + P(C_{\max})(b_i + b_j).$$

The tree that the chromosome represents is then found by applying Prim's algorithm (Gibbons 1985) to find a minimal-spanning tree (MST) over the nodes using the biased cost matrix. Finally, this MST is evaluated using the original cost matrix to determine the tree's fitness for the OCSTP.

This seemed sufficient at first, but we found that it did not cover the space of all trees in certain cases (Palmer 1994a). Although the kinds of tree not representable by this encoding were not good solutions to the OCSTP, in the interest of producing a generally useful tree representation we modified the representation to include link biases as well.

In this representation, the chromosome has biases for the N nodes and each of the $N(N-1)/2$ links, for a total of $N(N+1)/2$ biases. The GA itself has two additional parameters, P_1 and P_2, for use as multipliers (along with $C_{\max}$) on the link and node biases, respectively. The cost matrix is then biased by both of these values:

$$C'_{ij} = C_{ij} + P_1(C_{\max})b_{ij} + P_2(C_{\max})(b_i + b_j).$$

This version of the representation can encode any tree, T, given suitable values of the b_i (Palmer 1994a). This is easily seen by observing that we can set $b_i = 0$, for all i, and

$$b_{ij} = \begin{cases} 0 & \text{for } (i, j) \in T \\ M & \text{otherwise} \end{cases}$$

where M is larger than the maximum value of C_{ij}. The parameters P_1 and P_2 are fixed for a single GA experiment. For the OCSTP, we found that the b_{ij} biases were unimportant and so we ran our subsequent experiments with $P_1 = 0$. For our experiments we represented the b_i, b_j, and b_{ij} using eight bits, thus allowing the biases to take on values in the range $[0, 255]$.

We made several runs using this new representation and found it to be quite effective. We used the GAucsd v1.4 (Schraudolph and Grefenstette 1990) version of Grefenstette's Genesis (Grefenstette 1987) system for all of the results presented here. With only a little experimentation we found a single set of GAucsd parameters consistently found excellent solutions. The parameters that we chose were:

Population	100	σ Scaling	1.0
Crossover	0.6	P_1	0.0
Mutation	0.01	P_2	1.0
Gen Gap	1.0	Generations	100.

G1.3.5 Evaluation of the new encoding

Once we had established guidelines for the parameters best suited for our link-and-node-biased (LNB) GA for the OCSTP, we made several runs of the GA for five OCSTP problems with $6, 12, 24, 28,$ and 98

nodes. Since genetic algorithms are randomized procedures, we averaged several runs for each problem to obtain our results. We evaluated the effectiveness of the LNB GA in several ways. We first compared the results to a purely random search. We then compared the GA's performance to that of two good, known heuristic algorithms. Next, we changed the characteristics of the underlying network to strongly encourage a two-level, multiple-star, network in order to see how well the two techniques would adapt. Finally, we tested the adaptability of the GA further by applying it to an entirely different optimization problem.

G1.3.5.1 Random search comparison

We compared the distribution of solutions found by a purely random search to the distribution of the solutions to a 24-node problem found by the GA. As shown in figure G1.3.2, the genetic algorithm using 10 000 trials consistently found solutions superior to a random search of one million solutions by more than four standard deviations.

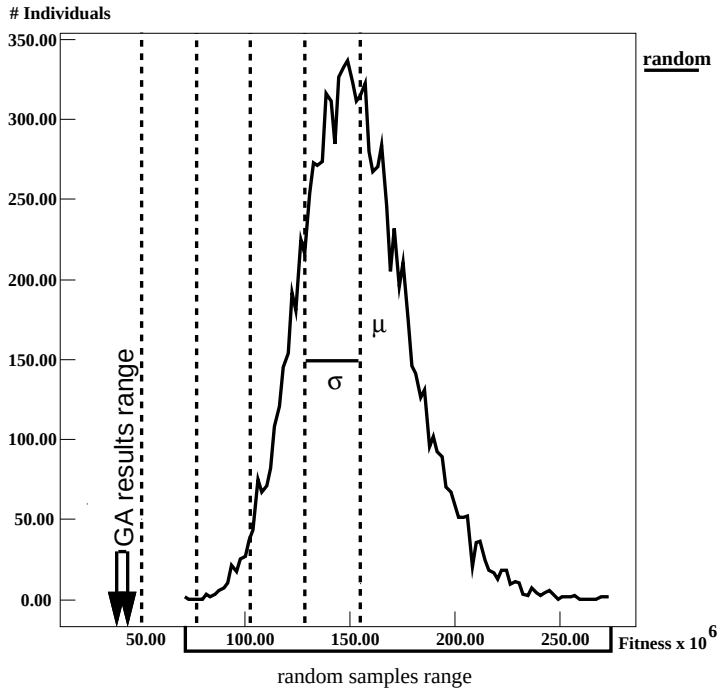


Figure G1.3.2. One million random samples for $N = 24$.

G1.3.5.2 Heuristic algorithm comparison

Since optimal solutions to the OCSTP are not generally known, we compared our GA with two good heuristics that have been used for the OCSTP in the past (Palmer and Kershenbaum 1994). The first heuristic begins by searching for the best tree that is a star on one node, then it searches for the best tree that is a connected pair of stars, and finally, starting with a MST, it tries to reduce the number of interior nodes by redirecting links to leaf nodes until it can improve no more. The heuristic returns the best tree found during these three steps. The second heuristic starts with a randomly generated tree and then uses a standard local exchange heuristic to try to improve it. The first heuristic simply runs to completion, while the second will continue to generate trees and try to improve them until a specified time constraint is exhausted.

When we compare the costs of the trees found by the LNB GA when using the parameters identified in section G1.3.4 to the trees found by the heuristics in tables G1.3.1 and G1.3.2, we find that the GA consistently found solutions as good as or better than those found by the heuristics. While the star search was considerably faster than the other heuristics, its results were several percent worse.

Table G1.3.1. Star search heuristic results comparison.

n	Star search	LNB genetic algorithm			
		Minimum	Average	Maximum	% gain
6	1.386×10^6	1.386×10^6	1.413×10^6	1.420×10^6	0
12	7.135×10^6	6.857×10^6	6.857×10^6	6.857×10^6	4.1
24	3.808×10^7	3.603×10^7	3.603×10^7	3.603×10^7	5.7
47	1.536×10^8	1.426×10^8	1.434×10^8	1.444×10^8	6.4–7.7
98	7.519×10^8	7.038×10^8	7.119×10^8	7.184×10^8	4.7–6.8

Table G1.3.2. Local exchange heuristic results comparison.

n	Local exchange		LNB genetic algorithm		
	Time	Minimum	Time	Minimum	% gain
6	1 min	1.386×10^6	1 min	1.420×10^6	0
12	3 min	6.857×10^6	3 min	6.857×10^6	0
24	12 min	3.664×10^7	11 min	3.603×10^7	1.7
47	1.2 days	1.478×10^8	58 min	1.426×10^8	3.7
98	4 days	7.331×10^8	347 min	7.038×10^8	4.2

Of course, halting a GA after some arbitrary number of evaluations may not allow the GA to complete its work. In our experiments, the LNB GA encoding easily provided very good solutions within the 10 000 evaluations budget we used.

G1.3.5.3 Adaptation to problem changes

Our goal was to design a GA that could be relied upon to produce very good solutions for the OCSTP even when faced with the kinds of change to the parameters of the problem that might cause a heuristic to break down. When someone designs a new heuristic, a reliable yardstick is needed in order to evaluate the heuristic. Older heuristics might be employed for this purpose, but they too are subject to changes in the problem definition and it may not be known whether they are finding really good solutions or just the best ones known. A GA that can adapt to changing problem parameters and still produce reliably good groups of solutions would be preferable. We investigated the adaptability of the LNB GA to two other tree problems which are described below.

Distribution network problem. In our original problem, the requirements between nodes were inversely proportional to their distance apart. While this is a reasonable model of some networks, we decided to change it to model another kind of network based on distribution centers. In the new problem, all of the requirements were regionalized into groups and were exclusively between leaf nodes and their 'center', with a token requirement between the centers. This gave rise to a problem where it was important for these centers to be interior nodes in the tree, a fact the heuristic was not taking into account.

For this new problem, the heuristic was unable to produce good solutions. The LNB GA, which makes use of no problem specific knowledge, adapted well to the problem and continued to produce good solutions. The results from running the heuristics and the LNB GA on the modified problem are shown in table G1.3.3.

Due to the much simplified traffic patterns in the modified problem, we were able to prove that the local exchange was finding the optimal solution (Palmer 1994a). As the above results show, the LNB GA was able to come within 0.5% of the optimal solution with no change to its control parameters.

Minimum-delay spanning tree problem. In another problem, a collection of local area networks (LANs) is given, along with the traffic requirements between all pairs of LANs including traffic within a LAN. The goal is to find a spanning tree connecting these LANs that minimizes the average network delay for

Table G1.3.3. Adaptability to the distribution network problem.

Technique	Best solution
Star search heuristic	3.210×10^6
Local exchange heuristic	2.173×10^6
LNB GA	2.183×10^6

Table G1.3.4. Direct comparison of GA and LX results to SA results.

n	D_{SA}	D_{GA}	$(D_{GA} - D_{SA})/D_{SA}$	D_{LX}	$(D_{LX} - D_{SA})/D_{SA}$
6	6.41	6.41	0%	6.41	0%
7	7.52	7.52	0%	7.52	0%
10	7.79	7.42	−4.75%	7.60	−2.4%
15	10.5	10.6	0.95%	10.9	3.8%
20	13.7	14.5	5.84%	13.8	0.7%
30	16.1	18.0	11.8%	15.6	−3.1%

all of the traffic requirements. In his dissertation (Ersoy 1992), Ersoy investigated this problem using the *simulated annealing* local search technique (Kirkpatrick *et al* 1983). It was interesting to compare the LNB GAs results for this problem to Ersoy's technique, which was specifically developed to address this problem. In addition, this problem requires the use of both the node and link biases to find good solutions. Table G1.3.4 contains a comparison of the results obtained by the GA and local exchange heuristic with the simulated annealing approach. The columns labeled D_{SA}, D_{GA}, and D_{LX} represent the minimum average network delay found by the simulated annealing, GA, and local exchange heuristic approaches, respectively.

D3.5

The quality of the GA results show that it was indeed able to adapt well to this quite different problem and to produce results that were comparable to those of an algorithm developed specifically for this problem. More importantly, the genetic algorithm was able to adapt to this problem without any retuning of its control parameters.

G1.3.6 Conclusions

We have defined a representation which satisfies most of the criteria given in section G1.3.2 and which can be used effectively to represent trees within a GA. It generates all trees and only trees. It requires only O(N) to convert to and from trees. It possesses good locality. Its only flaw is that it represents some trees more frequently than others; however this can be controlled parametrically. The LNB GA provided solutions to the three problems described that were reliably as good or better than those of the best available heuristics.

References

Ersoy C 1992 *Topological Design of Interconnected Local and Metropolitan Area Networks* Dissertation, Polytechnic University, Electrical Engineering Department, Brooklyn, New York

Gibbons A 1985 *Algorithmic Graph Theory* (New York: Cambridge University Press)

Gondran M and Minoux M 1984 *Graphs and Algorithms* (New York: Wiley)

Grefenstette J J 1987 *A User's Guide to Genesis 4.5* Technical Report, Navy Center for Applied Research in Artificial Intelligence, Naval Research Laboratory

Hu T C 1974 Optimum communication spanning trees *SIAM J. Comput.* **3** 188–95

Kershenbaum A 1992 *Telecommunications Network Design Algorithms* (New York: McGraw-Hill)

Kirkpatrick S, Gelatt C D and Vecci M P 1983 Optimization by simulated annealing *Science* **220** 671–80

Moon J W 1967 Various proofs of Cayley's formula for counting trees *A Seminar on Graph Theory* ed F Harary (New York: Holt, Rinehart and Winston) pp 70–8

Palmer C C 1994a *An Approach to a Problem in Network Design Using Genetic Algorithms* Dissertation, Polytechnic University, Computer Science Department, Brooklyn, New York

——1994b Representing trees in genetic algorithms *Proc. 1st IEEE Conf. on Evolutionary Computation (Orlando, FL, June 1993)* ed D B Fogel (Piscataway, NJ: IEEE) pp 379–84

Palmer C C and Kershenbaum A 1994 *Two Algorithms for Finding Optimal Communications Spanning Trees* IBM T J Watson Research Center Technical Report RC19394

Schraudolph N N and Grefenstette J J 1990 *A User's Guide to Gaucsd 1.4* Technical Report CS92-249, CSE Department, UC San Diego

G1.4 System identification using structured genetic algorithms

Hitoshi Iba

Abstract

This case study describes a new approach to system identification problems based on genetic programming (GP), and presents an adaptive system called STROGANOFF (structured representation on genetic algorithms for nonlinear function fitting). STROGANOFF integrates an adaptive search and a statistical method called group method of data handling (GMDH). More precisely, STROGANOFF consists of two processes: (i) the evolution of structured representations using a traditional genetic algorithm and (ii) the fitting of parameters of the nodes with a multiple-regression analysis. The fitness evaluation is based on a minimum-description-length (MDL) criterion. Our approach builds a bridge from traditional GP to a more powerful search strategy. In other words, we introduce a new approach to GP, by supplementing it with a local hill climbing. The approach is successfully applied to a time-series prediction.

G1.4.1 Introduction

System identification techniques are applied in many fields in order to predict the behaviors of unknown systems given input–output data (Astrom and Eykhoff 1971). This problem is defined formally in the following way. Assume that the single-valued output, y, of an unknown system behaves as a function of m input values: F1.4

$$y = f(x_1, x_2, \ldots, x_m). \tag{G1.4.1}$$

Given N observations of these input–output data pairs, that is,

Input				Output
x_{11}	x_{12}	...	x_{1m}	y_1
x_{21}	x_{22}	...	x_{2m}	y_2
		...		...
x_{N1}	x_{N2}	...	x_{Nm}	y_N

the system identification task is to approximate the true function f with $\bar{f}$. Once this approximate function $\bar{f}$ has been estimated, a predicted output $\bar{y}$ can be found for any input vector $(x_1, x_2, \ldots, x_m)$, that is,

$$\bar{y} = \bar{f}(x_1, x_2, \ldots, x_m). \tag{G1.4.2}$$

This $\bar{f}$ is called the 'complete form' of f.

STROGANOFF (i.e. structured representation on genetic algorithms for nonlinear function fitting) is aimed at solving system identification problems; it integrates an adaptive search of tree structures based on genetic programming (GP), and a local parameter tuning mechanism employing statistical search. STROGANOFF consists of two adaptive processes: (i) the evolution of structured representations, using a traditional genetic algorithm (GA); (ii) the fitting of parameters of the nodes with a multiple-regression analysis. The latter part is derived from a group method of data handling (GMDH) process, which is

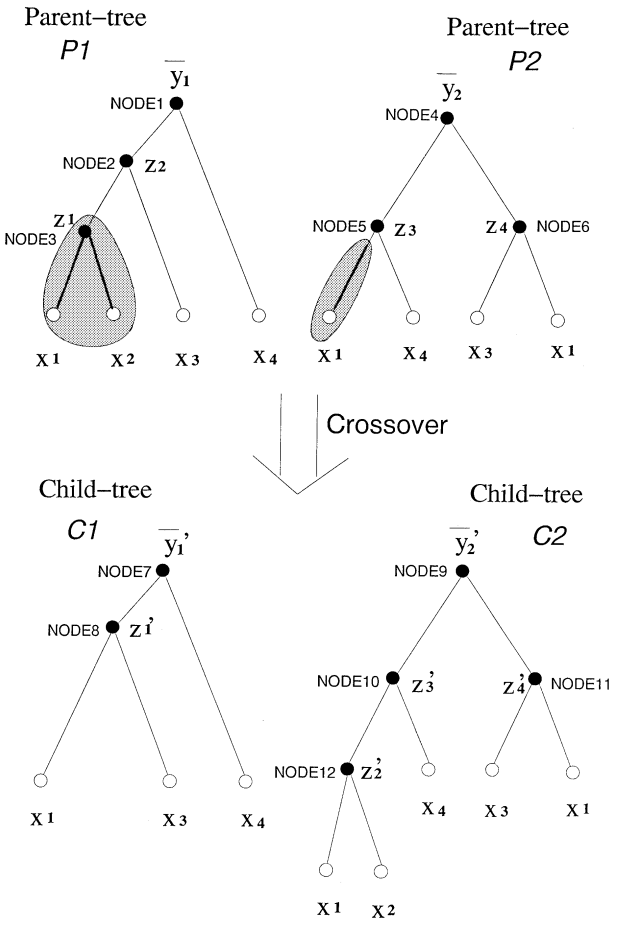


Figure G1.4.1. Crossover operation in STROGANOFF.

a multivariable analysis method, used to solve system identification problems (Ivakhnenko 1971). This method constructs a feedforward network, as it estimates the output function $\bar{f}$ (see Section F1.4 for details). The node transfer functions are simple (e.g. quadratic) polynomials of the two input variables, whose parameters are obtained using regression techniques. F1.4

G1.4.2 Design process

An example of a binary tree generated by STROGANOFF is shown in figure G1.4.1. For instance, the upper left parent tree (P_1) can be written as a (LISP) S-expression,

```
(NODE1
    (NODE2
        (NODE3 (x1) (x2))
        (x3)
    (x4)))
```

where x_1, x_2, x_3 and x_4 are the input variables. Intermediate nodes represent simple polynomial relationships between two descendant (lower) nodes. For the sake of simplicity, this case study assumes quadratic expressions for the intermediate nodes. Thus each node records the information derived by the following equations:

$$\text{NODE3} : z_1 = a_0 + a_1x_1 + a_2x_2 + a_3x_1x_2 + a_4x_1^2 + a_5x_2^2 \tag{G1.4.3}$$

$$\text{NODE2} : z_2 = b_0 + b_1z_1 + b_2x_3 + b_3z_1x_3 + b_4z_1^2 + b_5x_3^2 \tag{G1.4.4}$$

$$\text{NODE1} : \overline{y_1} = c_0 + c_1z_2 + c_2x_4 + c_3z_2x_4 + c_4z_2^2 + c_5x_4^2 \tag{G1.4.5}$$

where z_1 and z_2 are intermediate variables, and $\overline{y_1}$ is an approximation of the output, that is, the complete form. These equations are called 'subexpressions'. All coefficients $(a_0, a_1, \ldots, c_5)$ are derived from multiple-regression analysis using a given set of observations (see Spiegel 1975 for details). For instance, the coefficients a_i in (G1.4.3) are calculated using the following least-mean-square method. Suppose that N data triples (x_1, x_2, y) are supplied from observation:

$$\begin{matrix} x_{11} & x_{21} & y_1 \\ x_{12} & x_{22} & y_2 \\ & \ldots & \\ x_{1N} & x_{2N} & y_N. \end{matrix}$$

From these triples, an X matrix is constructed, such that

$$X = \begin{pmatrix} 1 & x_{11} & x_{21} & x_{11}x_{21} & x_{11}^2 & x_{21}^2 \\ 1 & x_{12} & x_{22} & x_{12}x_{22} & x_{12}^2 & x_{22}^2 \\ & & \ldots & & & \\ 1 & x_{1N} & x_{2N} & x_{1N}x_{2N} & x_{1N}^2 & x_{2N}^2 \end{pmatrix} \tag{G1.4.6}$$

where X is used to define a coefficient vector $\boldsymbol{a}$, given by

$$\boldsymbol{a} = (X'X)^{-1}X'\boldsymbol{y} \tag{G1.4.7}$$

where

$$\boldsymbol{a} = (a_0, a_1, a_2, a_3, a_4, a_5)' \tag{G1.4.8}$$

and

$$\boldsymbol{y} = (y_1, y_2, \ldots, y_N)'. \tag{G1.4.9}$$

X' is the transposed matrix of X. Note that all coefficients a_i are so calculated that the output variable z_1 approximates the desired output y. The other coefficients are derived in the same way.

We now consider the recombination of binary trees in STROGANOFF. Suppose two parent trees P_1 and P_2 are selected for recombination (figure G1.4.1). Besides the above equations, internal nodes record polynomial relationships as listed below:

$$\text{NODE5} \ : \ z_3 = d_0 + d_1x_1 + d_2x_4 + d_3x_1x_4 + d_4x_1^2 + d_5x_4^2 \tag{G1.4.10}$$

$$\text{NODE6} \ : \ z_4 = e_0 + e_1x_3 + e_2x_1 + e_3x_3x_1 + e_4x_3^2 + e_5x_1^2 \tag{G1.4.11}$$

$$\text{NODE4} \ : \ \overline{y_2} = f_0 + f_1z_3 + f_2z_4 + f_3z_3z_4 + f_4z_3^2 + f_5z_4^2. \tag{G1.4.12}$$

Suppose z_1 in P_1 and x_1 in P_2 (shaded portions in figure G1.4.1) are selected as crossover points in the respective parent trees. This gives rises to the two child trees C_1 and C_2 (lower part of figure G1.4.1). The internal nodes represent the following relations:

$$\text{NODE8} \ : \ z_1' = a_0' + a_1'x_1 + a_2'x_3 + a_3'x_1x_3 + a_4'x_1^2 + a_5'x_3^2 \tag{G1.4.13}$$

$$\text{NODE7} \ : \ \overline{y_1'} = b_0' + b_1'z_1' + b_2'x_4 + b_3'z_1'x_4 + b_4'z_1'^2 + b_5'x_4^2 \tag{G1.4.14}$$

$$\text{NODE12} \ : \ z_2' = c_0' + c_1'x_1 + c_2'x_2 + c_3'x_1x_2 + c_4'x_1^2 + c_5'x_2^2 \tag{G1.4.15}$$

$$\text{NODE10} \ : \ z_3' = d_0' + d_1'z_2' + d_2'x_4 + d_3'z_2'x_4 + d_4'z_2'^2 + d_5'x_4^2 \tag{G1.4.16}$$

$$\text{NODE11} \ : \ z_4' = e_0' + e_1'x_3 + e_2'x_1 + e_3'x_3x_1 + e_4'x_3^2 + e_5'x_1^2 \tag{G1.4.17}$$

$$\text{NODE9} \ : \ \overline{y_2'} = f_0' + f_1'z_3' + f_2'z_4' + f_3'z_3'z_4' + f_4'z_3'^2 + f_5'z_4'^2. \tag{G1.4.18}$$

Since these expressions are derived from multiple-regression analysis, we have the following equations:

$$z_2' = z_1 \tag{G1.4.19}$$

$$z_4' = z_4. \tag{G1.4.20}$$

Thus, when applying crossover operations, we need only derive polynomial relations for z_1', z_3', $\overline{y_1'}$, $\overline{y_2'}$. In other words, recalculation of the node coefficients for the replaced subtree (z_2') and nonreplaced subtree (z_4') is not required, which reduces much of the computational burden in STROGANOFF.

When applying mutation operations, we consider the following cases:

(i) A terminal node (i.e. an input variable) is mutated to another terminal node (i.e. another input variable).
(ii) A terminal node (i.e. an input variable) is mutated to a nonterminal node (i.e. a subexpression).
(iii) A nonterminal node (i.e. a subexpression) is mutated to a terminal node (i.e. an input variable).
(iv) A nonterminal node (i.e. a subexpression) is mutated to another nonterminal node (i.e. another subexpression).

STROGANOFF uses a fitness function based on minimum description length (MDL) for evaluating the tree structures. This fitness definition involves a tradeoff between certain structural details of the tree and its fitting (or classification) errors.

$$\text{MDL} = (\text{Tree\_Coding\_Length}) + (\text{Exception\_Coding\_Length}). \tag{G1.4.21}$$

The details of the MDL-based fitness function are described in Section C4.4. C4.4

The MDL fitness definition for our binary tree is defined as follows (Tenorio and Lee 1990):

$$\text{Tree\_Coding\_Length} = 0.5k \log N \tag{G1.4.22}$$

$$\text{Exception\_Coding\_Length} = 0.5N \log S_N^2 \tag{G1.4.23}$$

where N is the number of data pairs, S_N^2 is the mean square error, that is,

$$S_N^2 = \frac{1}{N} \sum_{i=1}^{N} |\overline{y_i} - y_i|^2 \tag{G1.4.24}$$

and k is the number of parameters of the tree; for example, the k-value for the tree P_1 in figure G1.4.1 is $6 + 6 + 6 = 18$ because each internal node has six parameters ($a_0, \ldots, a_5$ for NODE3 and so on).

The STROGANOFF algorithm is described below.

Input: $t_{\max}$, I, Pop_size
Output: x, the best individual ever found.

$t \leftarrow 0$;
{I *is a set of input variables (see equation (G1.4.1)).* NODE_2 *is a nonterminal node of 2-arity*}
$P(t) \leftarrow$ initialize(Pop_size, I, {NODE_2});
$F(t) \leftarrow$ evaluate($P(t)$, Pop_size);
$x \leftarrow a_j(t)$ **and** $Best\_so\_far \leftarrow \text{MDL}(a_j(t))$, where $\text{MDL}(a_j(t)) = \min(F(t))$;
{*the main loop of selection, recombination, mutation*}
while ($\iota(P(t), F(t), t_{\max}) \neq$ true) **do**
    for $i \leftarrow 1$ **to** $\frac{Pop\_size}{2}$ **do**
        {*select parent candidates according to the MDL values*}
        $Parent_1 \leftarrow select(P(t), F(t), Pop\_size)$;
        $Parent_2 \leftarrow select(P(t), F(t), Pop\_size)$;
        {*apply GP crossover operation, that is, swapping subtrees (figure G1.4.1)*}
        $a'_{2i-1}(t), a'_{2i}(t) \leftarrow$ GP_recombine($Parent_1$, $Parent_2$);
        {*apply GP mutation operation,*
        i.e. changing a node label and deleting/inserting a subtree.}
        $a''_{2i}(t) \leftarrow$ GP_mutate($a'_{2i}(t)$);
        $a''_{2i-1}(t) \leftarrow$ GP_mutate($a'_{2i-1}(t)$);
    od
    $P''(t) \leftarrow (a''_1(t), \ldots, a''_{Pop\_size}(t))$;
    $F(t) \leftarrow$ evaluate($P''(t)$, Pop_size);
    $tmp \leftarrow a''_k(t)$, where $\text{MDL}(a''_k(t)) = \min(F(t))$;
    if ($Best\_so\_far > \text{MDL}(a''_k(t))$)
        then $x \leftarrow tmp$ **and** $Best\_so\_far \leftarrow \text{MDL}(a''_k(t))$;
    $P(t+1) \leftarrow P''(t)$;
    $t \leftarrow t + 1$;
od
return (x);

```
    {terminate if more than t_max generations are over}
1   ι(P(t), F(t), t_max) :
2       if (t > t_max)
            then return true;
            else return false;

    {initialize the population randomly}
1   initialize(Pop_size, T, F):
2       for i ← 1 to Pop_size do
            generate a tree a_i randomly,
                where the terminal and nonterminal sets are T and F.
        od
        return (a_1, ..., a_Pop_size);

    {evaluate of a population of size Pop_size}
1   evaluate(P(t), Pop_size):
2       for i ← 1 to Pop_size do
            {calculate ((G1.4.24))}
            GMDH_Process(a_i);
            S_N^2(a_i) ← the mean square error of a_i;
            {calculate equations (G1.4.21), (G1.4.22), and (G1.4.23)}
            MDL(a_i) ← Tree_Coding_Length(a_i) + Exception_Coding_Length(a_i);
        od
        return (MDL(a_1), ..., MDL(a_Pop_size));

    {execute the GMDH process}
1   GMDH_Process(a):
2       nd ← the root node of a;
3       if (nd is a terminal node)
            then return;
        {if the node coefficients of nd are already derived, then return}
4       if (Coeff(nd) ≠ NULL)
            then return;
5       nl ← left_child(nd);
6       nr ← right_child(nd);
7       GMDH_Process(nl);
8       GMDH_Process(nr);
9       Coeff(nd) ← Mult_Reg(nl, nr);
        return;

    {execute the multiple-regression analysis}
1   Mult_Reg(n1, n2):
        Assume n1 is the first variable and n2 is the second variable.
        For instance, x_1 ← n1, x_2 ← n2 for (G1.4.3)
        Derive and return the fitting coefficients, that is, (G1.4.8)
        return;
```

In the GMDH_Process called by the evaluate routine, the coefficients of the child trees are recalculated using the multiple regressions. However, this recalculation is performed only on intermediate nodes, upon whose descendants crossover or mutation operators were applied (see the fourth lines in GMDH_Process). Therefore, the computational burden of the GMDH process is expected to be reduced as the generations proceed. As can be seen, lines from 6 to 7 in the STROGANOFF algorithm follow traditional GP, whereas GMDH_Process is the new local hill climbing procedure, which will be discussed later.

G1.4.3 Development and implementation

STROGANOFF is grounded on GP search. The system was firstly implemented in LISP, later in C, on a Sun4 (or Sparc10) workstation. STROGANOFF consumes much of the computational time to perform the multiple-regression analysis. However, we believe that parallelizing STROGANOFF (i.e. both the GP process and the statistical process) will lead to a reduction of the computational burden. We are currently working on this topic as part of the Real World Computing (RWC) project.

G1.4.4 Results

The Mackey–Glass differential equation

$$\frac{\mathrm{d}x(t)}{\mathrm{d}t} = \frac{ax(t-\tau)}{1+x^{10}(t-\tau)} - bx(t) \tag{G1.4.25}$$

is used for time-series prediction problems, where $a = 0.2, b = 0.1$ and $\tau = 17$ (figure G1.4.2(a)). This is a chaotic time series with a strange attractor of fractal dimension of approximately 3.5 (Tenorio and Lee 1990).

In order to predict this series, the first 100 points (i.e. the values of $x(1), \ldots, x(100)$) were given to STROGANOFF as training data. The aim was to obtain a prediction of $x(t)$ in terms of M past data:

$$x(t) = f(x(t-1), x(t-2), \ldots, x(t-M)). \tag{G1.4.26}$$

The parameters for STROGANOFF were as follows:

Population size	:	60
Crossover probability	:	0.6
Mutation probability	:	0.0333
Terminal nodes	:	$\{x(t-1), x(t-2), \ldots, x(t-10)\}$.

We used ten past data (i.e. $M = 10$) for simplicity.

Figures G1.4.2(b) and (c) are the time series predicted by STROGANOFF (generations 233 and 1740 respectively). Note that the selection process of STROGANOFF is based on the MDL-value, and not on the raw fitness (i.e. the error-of-fit ratio). The resulting structure of figure G1.4.2(c) was as follows:

```
(NODE95239 (7)
   (NODE95240
      (NODE95241
         (NODE95242
            (NODE95243
               (NODE95244
                  (8)
                  (NODE95245
                     (8)
                     (NODE95130 (2) (3))))
               (NODE95173
                  (10)
                  (NODE95174
                     (NODE95175 (4) (1))
                     (5))))
            (5))
         (6))
      (NODE95178 (NODE95179 (8) (3)) (10))))
```

where (i) represents $x(t-i)$. Some of the node coefficients are shown in table G1.4.1.

Note that in figure G1.4.2(c) the prediction at the 1740th generation fits the training data almost perfectly. We then compared the predicted time series with the testing time series (i.e. $x(t)$ for $t > 100$). This also produced good results (compare figure G1.4.2(a) and figure G1.4.2(c)). The mean squared errors for this period are summarized in table G1.4.2.

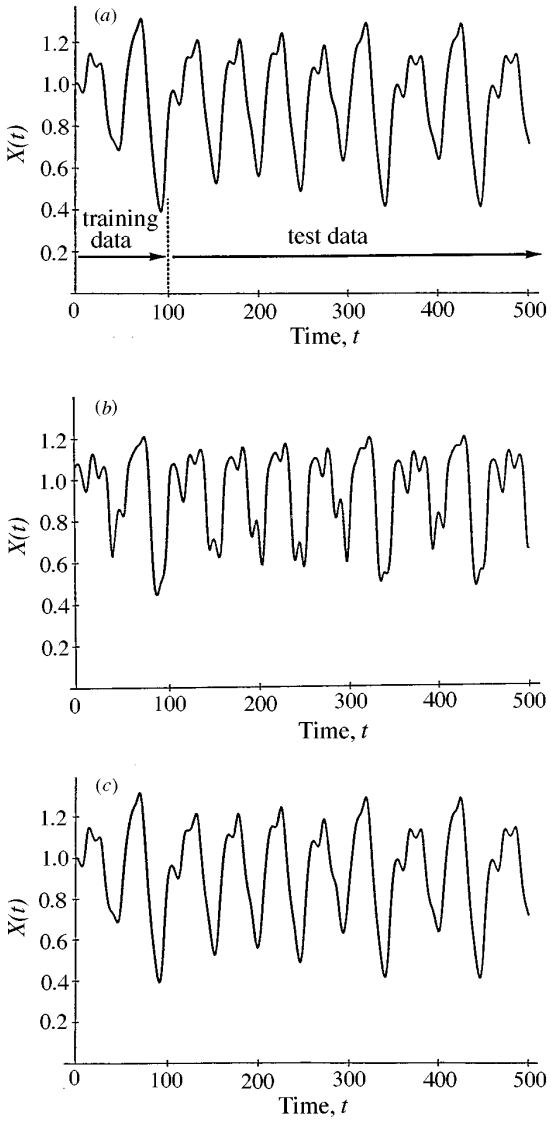


Figure G1.4.2. (*a*) Chaotic time series. (*b*) Prediction at 233rd generation. (*c*) Prediction at 1740th generation.

Table G1.4.1. Node coefficients.

	NODE95239	NODE95240	NODE95179
a_0	0.093	−0.090	0.286
a_1	0.133	1.069	−0.892
a_2	0.939	−0.051	1.558
a_3	−0.029	1.000	1.428
a_4	0.002	−0.515	−0.536
a_5	−0.009	−0.421	−0.844

Table G1.4.2. Mean square errors (STROGANOFF).

Generation	Training data	Testing data	MDL
233	0.012 15	0.012 61	−192.86
1740	4.70×10^{-6}	5.06×10^{-6}	−433.79

The MDL value (i.e. fitness) of this tree is given as follows:

$$\text{MDL fitness} = 0.5k \log N + 0.5N \log S_N^2 \tag{G1.4.27}$$

$$= 0.5 \times (6 \times 13) \times \log 100 + 0.5 \times 100 \times \log(4.70 \times 10^{-6}) \tag{G1.4.28}$$

$$= -433.79 \tag{G1.4.29}$$

where the number of training data (i.e. N) is 100, and the MSE (i.e. S_N^2) is 4.70×10^{-6}. Since the number of intermediate nodes is 13, the k-value is roughly estimated as 6×13, because each internal node has six parameters.

G1.4.5 Comparison with a traditional genetic program

Traditional GP has also been applied to the prediction task (Oakley 1994). In order to compare the performance of STROGANOFF, we applied a traditional GP system 'sgpc1.1' (a simple genetic programming in C written by Walter Alden Tackett) to the same chaotic time series (i.e. Mackey–Glass equation). For the sake of comparison, all the parameters chosen were the same as those used in the previous study (Oakley 1994, p 380, table 17.3), except that the terminal set consisted of ten past data for the short-term prediction (see table G1.4.3).

Table G1.4.4 gives the results of the experiments, showing the mean square error of the best performance over 20 runs. For the sake of comparison, we also list the results given by STROGANOFF. The numbers of individuals to be processed are shown in the third column (i.e. #Pop $\times$ Gen.).

The trees resulting from traditional GP are as follows:

```
<<Generation 67>>
(
  (SIN
    (+
      (
        X(t-8)
        X(t-3))
      X(t-9)))
  (
    (
      X(t-5)
      X(t-1))
    (
      X(t-4)
      (EXP10
        X(t-7)))))
<<Generation 87>>
(-
  (+
    X(t-1)
    X(t-1))
  X(t-2))
```

The experimental results show that the traditional GP suffers from overgeneralization, in the sense that the mean square error of the test data (i.e. 1.50×10^{-4}) is much worse than that of the training data (i.e. 6.50×10^{-6}). This may be caused by the fact that traditional GP has no appropriate criterion (such as MDL for STROGANOFF) for evaluating the tradeoff between the errors and the model complexities (i.e. the description length of S-expressions).

Another disadvantage of traditional GP is due to the mechanisms used to generate constants. In traditional GP, constants are generated randomly by initialization and mutation. However, there is no tuning mechanism for the generated constants. This may degrade the search efficiency, especially in the case of time-series prediction tasks, which require a fine tuning of the fitting coefficients, so the number of processed individuals for the same quality of solution is much greater than for STROGANOFF (see the third column in table G1.4.4).

Comparative studies of STROGANOFF with other optimization techniques are given by Iba and Sata (1994b).

Table G1.4.3. GP parameters (predicting the Mackey–Glass equation).

Objective	Predict next data $X(t)$ in Mackey–Glass mapping series
Terminal set	Time-embedded data series from $t = 1, 2, \ldots, 10$, i.e. $\{X(t-1), X(t-2), \ldots, X(t-10)\}$, with a random constant
Function set	$\{+, -, \times, \%, \text{SIN}, \text{COS}, \text{EXP10}\}$
Fitness cases	Actual members of the Mackey–Glass mapping ($t = 1, 2, \ldots, 500$)
Raw fitness	Sum over the fitness cases of squared error between predicted and actual points
Standardized fitness	Same as raw fitness
Parameters	$M = 5000, \quad G = 101$
Maximum depth of new individuals	6
Maximum depth of mutant subtrees	4
Maximum depth of individuals after crossover	17
Fitness-proportionate reproduction fraction	0.1
Crossover at any point fraction	0.2
Crossover at function points fraction	0.7
Selection method	Fitness-proportionate
Generation method	Ramped half-and-half

Table G1.4.4. Mean square errors (GP against STROGANOFF).

System	Gen.	#Pop × Gen.	Training data	Testing data
STROGANOFF	233	13 980	0.012 15	0.012 61
	1740	104 440	4.70×10^{-6}	5.06×10^{-6}
sgpc1.1 (GP)	67	325 000	9.62×10^{-4}	2.08×10^{-3}
	87	435 000	6.50×10^{-6}	1.50×10^{-4}

G1.4.6 Summary

We applied STROGANOFF to several problems such as other time-series predictions, pattern recognitions, 0–1 optimizations and temporal data processing. The results obtained were satisfactory. Detailed discussions are given by Iba *et al* (1993, 1994a, b, 1995) and Iba and Sato (1994a, b). We feel that STROGANOFF searches in an efficient manner because it is grounded on a GP approach, which integrates a statistical regression method and an adaptive search strategy. The advantages of our system identification approach to GP are summarized as follows:

(i) Analog (polynomial) expressions are complemented with digital (symbolic) semantics.
(ii) MDL-based fitness evaluation works well for tree structures, which controls GP-based tree search.
(iii) GP operation (i.e. crossover and mutation) is guided adaptively (see Iba and deGaris 1996 for details).

We have introduced a way to modify trees, by integrating node coefficient tuning and traditional GP recombination, that is by supplementing GP with a local hill climbing approach. Local hill climbing search uses local parameter tuning (of the node functionality) of tree structures, and works by discovering useful substructures in STROGANOFF trees. Our proposed augmented GP paradigm can be considered schematically in several ways:

augmented GP = global search + local hill climbing search
= structured search + parameter tuning of node functionalities.

The local hill climbing mechanism uses a type of relabeling procedure, which finds a locally (if not globally) optimal assignment of nodes for an arbitrary tree. The term ‘label’ is used to represent the

information (such as a function or polynomial) at a nonterminal node. Therefore, speaking generally, our new approach can be characterized as

augmented GP = traditional GP + relabeling procedure.

The augmented GP algorithm is described below:

Step 1 Initialize a population of tree expressions.
Step 2 Evaluate each expression in the population.
Step 3 Create new expressions (children) by mating current expressions. Apply mutation and crossover to the parent tree expressions.
Step 4 Replace the members of the population with the child trees.
Step 5 A local hill climbing mechanism (called 'relabeling') is executed periodically, so as to relabel nodes of the trees of the population.
Step 6 If the termination criterion is satisfied, then halt; else go to Step 2.

As can be seen, Steps 1–5 follow traditional GP, where Step 4 is the new local hill climbing procedure. In our augmented GP paradigm, the traditional GP representation (i.e. the terminal and nonterminal nodes of tree expressions) is constrained so that our new relabeling procedure can be applied. The sufficient condition for this applicability is that the designed representation have the property of 'insensitivity' or 'semantic robustness', that is, changing a node of a tree does not affect the semantics of the tree. In other words, the GP representation is determined by the choice of the local hill climbing mechanism.

In this case study, we have chosen a GMDH algorithm as the relabeling procedure for system identification problems. We are currently pursuing other relabeling procedures for various kinds of problem domain. For instance, in the articles by Iba *et al* (1994b, 1995), we have chosen two vehicles to perform the relabeling procedure, known respectively as ALN and the error propagation. The characteristics of these resulting GP variants are summarized in table G1.4.5.

Table G1.4.5. Properties of GP variants.

	Boolean GP	STROGANOFF	$\Re$-STROGANOFF
Problem domain	Boolean concept formation	system identification	temporal data processing
Tree type	binary tree		network
Terminal nodes	input variables their negations	input variables	input variables
Nonterminal nodes	AND, OR, LEFT, RIGHT	polynomial relationships	polynomial relationships, memory
Relabeling process	ALN	GMDH	error propagation

References

Astrom K J and Eykhoff P 1971 System identification, a survey *Automatica* **7** 123–62

Iba H, Kurita T, deGaris H and Sato T 1993 System identification using structured genetic algorithms *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufman) pp 279–86

Iba H, deGaris H and Sato T 1994a *System Identification Approach to Genetic Programming* Electrotechnical Laboratory Report ETL-TR94-2; *Proc. IEEE World Congress on Computational Intelligence (Orlando, FL, June 1994)* (Piscataway, NJ: IEEE) pp 401–6

——1994b Genetic programming using a minimum description length principle *Advances in Genetic Programming* ed K E Kinnear Jr (Cambridge, MA: MIT Press) pp 265–84

Iba H and Sato T 1994a Genetic programming with local hill-climbing *Parallel Problem solving from Nature—PPSN III (Proc. Int. Conf. on Evolutionary Computation and 3rd Conf. on Parallel Problem Solving from Nature, Jerusalem, October 1994)* (*Lecture Notes in Computer Science 866*) ed Yu Davidor, H-P Schwefel and R Männer (Berlin: Springer) pp 302–411

——1994b *Genetic Programming using a System Identification Approach* Electrotechnical Laboratory Report ETL-TR94-11

Iba H, deGaris H and Sato T 1995 Temporal data processing using genetic programming *Proc. 6th Int. Conf. on Genetic Algorithms (Pittsburgh, PA, July 1995)* ed L J Eshelman (San Mateo, CA: Morgan Kaufmann) pp 279–86

Iba H and deGaris H 1996 Extending genetic programming with recombinative guidance *Advances in Genetic Programming 2* ed P Angeline and K Kinnear (Cambridge, MA: MIT Press)

Ivakhnenko A G 1971 Polynomial theory of complex systems *IEEE Trans. Syst. Man Cybernet.* **SMC-1** no 4

Oakley H 1994 Two scientific applications of genetic programming: stack filters and nonlinear equation fitting to chaotic data *Advances in Genetic Programming* ed K E Kinnear Jr (Cambridge, MA: MIT Press)

Spiegel M R 1975 *Theory and Problems of Statistics* (New York: McGraw-Hill)

Tenorio M F and Lee W 1990 Self-organizing network for optimum supervised learning *IEEE Trans. Neural Networks* **NN-1** 100–9

G1.5 Comparison of a compiling genetic programming system versus a connectionist approach

Peter Nordin

Abstract

This case study briefly presents the compiling genetic programming method and evaluates its performance against a neural network. Most genetic programming approaches use a technique where a problem specific language is executed by an interpreter. The individual code segments in the population are decoded at run time by a virtual machine. The disadvantage of this paradigm is that interpreting the program involves a large overhead. We have evaluated the idea of using the lowest-level native binary machine code as the individuals in the population. There is no intermediate language nor any interpreting steps. The genetic program that administers these machine code segments is written in C. The algorithm is steady state and uses a small tournament for selection. This approach has enhanced performance by up to 2000 times compared to a conventional system in an interpreting language. The increased performance is tested on a problem of symbolic regression of a classifier function in machine code. We evolve a machine code program that classifies Swedish words into nouns and non-nouns by spelling only. We compare the compiling genetic programming system (CGPS) with a neural network performing the same task. In our example, the results show superior performance of the CGPS compared to the connectionist approach. While the classification and generalization capabilities are equal, the training time is more than 200 times faster, the classification time 500 times faster, and the memory requirements at least ten times lower with the CGPS, as compared with the neural network.

G1.5.1 Project overview and introduction

Genetic programming uses the mechanisms behind natural selection for evolution of computer programs. B1.5.1 Instead of a human programmer programming the computer, the computer can self-modify, through genetic operators, a population of programs in order to finally generate a program that solves the defined problem (Koza 1992). This technique, like other adaptive techniques, has applications in problem domains where theories are incomplete and insufficient for the human programmer or when there is insufficient time or resources available to allow for human programming.

The set of practically solvable tasks is highly related to the efficiency of the algorithm and implementation. It is therefore important to minimize the overhead involved in executing the individuals in a genetic programming system. In this article we present a variant of genetic programming that enables a very efficient implementation.

Most genetic programming approaches use a technique where a problem specific language is executed by an interpreter. The individuals in the population are decoded at run time by a virtual machine. The data structures in the programs often have the form of a tree. This solution gives good flexibility and the ability to customize the language depending on the constraints of the problem at hand. The disadvantage of this paradigm is that interpreting the program involves a large overhead. There is also a need to define more complicated genetic operators, which also decreases performance. Often the complete system and

the genetic operators themselves are written in an interpreting language like LISP (Koza 1992). This reduces performance in most hardware environments. Recently some systems have been presented written in compiler languages like C or C++, parsing structures equivalent to the programs used in a LISP implementation (Keith and Martin 1993). This gives increased performance while it preserves the ability to be flexible with the representation and selection of a problem specific function set. Still there is a need to interpret the programs in the population that involves overheads both in execution time and memory consumption.

We have evaluated the idea of using the lowest-level binary machine code as the 'programs' in the population. Every individual is a piece of machine code that is called and manipulated by the genetic operators. There is no intermediate language or interpreting part of the program. The machine code program segments are invoked with a standard C function call. The system performs repeated type cast between pointers to arrays for individual manipulation and pointers to functions for the execution of the programs and evaluation of the fitness of the individuals. Legal and valid C-functions are put together, at run time, directly in memory by the genetic algorithm. This is a 'compiling' genetic programming system (CGPS).

G1.5.2 Design process: the compiling genetic programming system (CGPS)

We call our approach 'compiling' as the system generates binary code from the example set and there are no interpreting steps. The idea is to use the real machine instead of a virtual machine and the hypothesis is that the loss in flexibility will be well compensated for by increased efficiency.

G1.5.2.1 Structure of a machine code function callable as a C-function

The individuals in the population consist of machine code sequences resembling a standard C-function as stated above. The two biggest differences between a canonical genetic programming system and the CGPS is that the CGPS has a linear genome and a crossover operator working on linear structures. These differences are motivated by the representation of a function in binary code. Figure G1.5.1 illustrates the structure of a function in machine code. This structure is fairly generic for different types of compiler and hardware architecture.

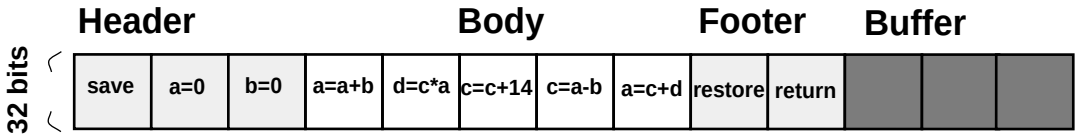


Figure G1.5.1. Structure of program individual.

The function code consists of four major parts:

- The *header* deals with the administration necessary when entering a function. This normally means manipulation of the stack, for instance obtaining the arguments for the function from the stack. There could also be some processing to ensure consistency of processor registers. This section is often constant and can be added at the beginning of the initialization of the individual machine code segments in the population. The mutation and crossover operators must be prevented from changing this part during evolution.
- The *footer* is similar to the header but performs the operations in the opposite order and 'cleans up' after the function call. The footer must also be protected from change by the genetic operators. The return instruction forces the system to leave the function and return program control to the calling procedure. If variable-length programs are desired, then the return operator could be allowed to move within a range defining minimum and maximum program size.
- The *function body* consists of the actual program that evaluates the function.
- A *buffer* is reserved at the end of each individual to allow for length variations.

G1.5.2.2 Genetic operators

The evolutionary algorithm has the following three operators.

- We used an algorithm with a steady-state *tournament selection* (Reynolds 1992). C2.3
- The mutation operator has two parts, one working on the operator and one on the operand. The one working on the operand randomly changes 1 bit of the operand if certain criteria are fulfilled. Before this can be done a check has to be made to see whether this instruction has an operand. The other part working on the operator changes it to a member of the set of approved instructions to assure that there will be no jumps, illegal instructions, bus errors, loops or the like. It also assures some arithmetic consistency, where for instance division by zero is prevented.
- We have evaluated two methods for crossover.
  - The first method is similar to the standard *uniform crossover* used in genetic algorithms. Here it works down to the bit level for operands, but preserves the integrity of instructions by preventing a crossover in the operand part of an instruction. The technique emulates a crossover on one long continuous bitstring. For more details on this protected crossover operator see the article by Nordin (1994). C3.3
  - The second method operates on variable-length individuals. Crossover is only allowed between instructions at 32-bit intervals in the binary string. Crossover performs a cut and splice similar to the operator of *messy genetic algorithm systems* (Goldberg *et al* 1989). For more details on this CGPS crossover variant see the article by Nordin and Banzhaf (1995). B1.2

  In the case study below we use the first crossover method with a fixed-length individual.

G1.5.2.3 A genetic programming system for heuristic classification

We have applied our compiling genetic programming system to a heuristic classification task. The reason for choosing this task is a combination of application interest and the ability to compare the system to a more established classification paradigm, neural networks.

The goal is to evolve a classification function in binary machine code and be able to classify objects from a domain learned by supervised learning. The result is intended to be a function that is able to generalize and classify examples that did not appear in the training set, but that presumably had something in common with the examples in the training set. The machine code functions, the individuals in the population, take a 32-bit integer as input and always return a 32-bit output.

There is a training set divided into positive and negative examples. The goal of the function is that if it is called with a positive example as argument it then should return a value as low as possible, ideally zero. Similarly, when the function is called with a negative example, it should return the highest possible value. The output from the function is masked, that is, it only returns the 16 least-significant bits. In reality the highest output from the function individuals is $2^{16} - 1 = 65\,535$. This is thus the ideal output for a negative example. The fitness for a single positive example is simply the value returned, while the fitness for a negative example is 65 535 subtracted from the value returned. The fitness function for an individual program is the sum of these single fitness case values over the complete training set.

This approach could be applied to a number of problem domains and we have chosen an initial evaluation of the task of machine parsing of natural language.

G1.5.3 Comparison between the CGPS and a neural network

G1.5.3.1 The sample problem

When we looked for a sample problem suitable for the comparison of CGPS and a connectionist paradigm, we wanted a problem with few known rules and with fuzzy boundaries, that could not be labeled as a trivial problem. We searched for a domain that had complex dependences where the optimum solution was unknown, and we wanted it to have some practical relevance and application. Finally we chose the task of classifying words into noun and non-noun using the spelling of the words only and not relating the words to any context or other information. It is not the intention here to make a complete investigation and evaluation of this problem space, nor to make a complete comparison between CGPS and neural networks. This test gives an illustration of the potential of a CGPS and its relation to a more established technique for heuristic classification.

G1.5.3.2 The neural network

The neural network is a three-layer perceptron network. It has six input nodes corresponding to the letters in the words. It has eight hidden nodes fully connected with the input nodes and one output node. The nodes have a sigmoid activation function and the weights are trained by the backpropagation delta rule with momentum (Rumelhart *et al* 1986). Table G1.5.1 below shows a summary of the parameters used for the training algorithm.

Table G1.5.1. Parameters for the neural network algorithm.

Low end initialization range	−0.1
High end initialization range	0.1
Success criterion	0.3
Momentum	0.9
Weight learning rate	0.1
Bias learning rate	0.1

G1.5.3.3 The training set

The training set consists of 2100 Swedish words. These words were sequentially collected from short newspaper articles. Very little processing was done after extracting them from these articles. Proper nouns were deleted but duplicates and homonyms were kept. The original order of the words was kept. This method is used to mimic the environment of a real application. From this large set of words, 21 training sets were extracted, each consisting of 100 words, 50 nouns and 50 non-nouns. Every training set thus consisted of one negative training set and one positive training set of 50 words each. This ASCII information then had to be transformed and coded into numerical form to fit the requirements of the CGPS and the neural network. The coding was performed in two different ways, each reflecting the specific needs of one of the two paradigms.

G1.5.3.4 Coding of words for the genetic programming system

As stated above the individual functions in the population of the genetic programming system only take a 32-bit integer as input. To squeeze the words into this compressed format we use a representation of 5 bits per letter, which gives a total of six letters maximum in every word to be used as input. If we have a word longer than six letters we use the last six letters only. We suspect that most information about a word's class is manifested in its last letters.

Table G1.5.2. Code numbers for letters in the Swedish alphabet.

Space = 0	A = 1	Å = 2	O = 3	Ö = 4
Ä = 5	E = 6	I = 7	U = 8	Y = 9
B = 10	P = 11	D = 12	F = 13	G = 14
H = 15	J = 16	C = 17	K = 18	L = 19
M = 20	N = 21	Q = 22	R = 23	S = 24
T = 25	V = 26	X = 27	Z = 28	

For the coding of single letters into 5 bits we apply a straightforward method where every letter is given a number according to its place in a list of the 28 letters in the Swedish alphabet. This list is composed to reflect some similarities between letters: vowels are for instance placed in the beginning of the list (see table G1.5.2).

These codes are then combined to 32 bits, giving 5 bits to each letter, where the five least-significant bits come from the last letter of the encoded word. More formally this can be represented as follows:

$$C = \sum (32 \times 2^i l_i). \tag{G1.5.1}$$

Here C represents the code number to be fed into the individual function in the population, and l_i is the number from table G1.5.2 where l_0 corresponds to the last letter of the word to be encoded. The most common Swedish word 'och' meaning 'and' will be coded to the number 3631 which thus frequently occurs in the non-noun training set.

G1.5.3.5 Coding of words for the neural network

Coding the words for the neural network is somewhat less complex. The network has six input nodes one for each letter. The input to each node is computed by the formula

$$C_i = 0.033 l_i \tag{G1.5.2}$$

where C_i are the values fed into the input nodes. Note that the coding for the neural network is slightly advantageous since the coding divides the words by letter boundaries while the coding for the CGPS make no such distinction.

G1.5.3.6 Training method

The training and comparison was performed as follows. The 21 training sets were iterated through and used to train the CGPS and the neural network. The curves of the fitness function and the mean square error were plotted and the point where the derivative of the curves flattens out was measured.

When the population has stabilized, the best individual is chosen and the number of correct classifications is measured. The program individual is then tested on one of the other 20 training sets which it has not been trained on. This gives a value of generalization capability. The resulting neural network is also applied to an unseen validation set to measure generalization for this paradigm. A mean value over all 21 training sets is calculated for classification in the current training set and in a fresh unseen validation set.

G1.5.4 Development and implementation

The main implementation advantage of our approach to genetic programming is that it enables the direct manipulation of binary individuals. This gives a huge execution speed advantage over interpreting methods. Benchmark tests between the CGPS and an equivalent genetic programming system written in the interpreting language LISP show that the compiling system executes up to 2000 times faster than the interpreting system. In the same study we concluded that the CGPS is up to 100 times faster than an interpreting system written in C. The system is also very memory efficient in both kernel size and individual size. The kernel is about 30 kbytes while the memory consumption is about one byte per node (Nordin and Banzhaf 1995).

G1.5.4.1 The instruction set

To limit the search space and to avoid complex control mechanisms and thus achieve maximum performance we have restricted the set of machine instructions in this example. We use two processor registers and only two address mode types. With these kinds of instruction it is possible to reach the maximum processor throughput. An average individual program consisting for instance of a dozen instructions of this type can be executed in one fifth of a microsecond on a modern desktop computer, and the task that the genetic programming aims at, for instance the classification, is thus performed in the same time. In this example, we have also constrained the problems to those that could be solved by a function taking an integer as argument and returning an integer. This however does not limit the problem space to mathematical problems as illustrated below. The operations used are multiplication, division, addition, subtraction, AND, OR, XOR, and shifts. No control instructions like jump instructions are thus allowed in this application which means that no loops can be formed. For further details on these instructions see Sun Microsystems (1990).

In this case study the instruction set is limited, but compiling genetic programming can be used with most program constructs, such as arithmetic operators, large indexed memory, automatic decomposition into subfunctions and subroutines (ADFs), conditional constructs, that is if–then–else, jumps, loop structures, recursion, protected functions, multiple-input and output structures, and string and list functions. Any

C-function can also be compiled and linked into the function set of the system. See the article by Nordin and Banzhaf (1995) for implementation details of the more advanced features.

G1.5.5 Results

The results shows that both paradigms have about the same ability to learn the training set: an average of 89% of the training examples were learned by the neural network compared to 86% for the CGPS. When confronted with words that it has not been trained on, the neural network correctly classifies 69%, on average. The CGPS classifies 72% correctly in the fresh validation set. These measurements are made with a population size of 4000 and an individual size of 12 instructions.

The average training time was 235 minutes for the neural network compared to approximately 1 minute for the CGPS. All times are clocked on a Sun Sparcstation 1+. The CGPS clearly outperforms the network paradigm with regards to training time in this problem domain.

The training algorithm looks at 4000 examples to arrive at the final network and the CGPS processes four million examples. In spite of this, the genetic system is much faster. One possible enhancement for the future could be to look at different strategies for how to present the training examples to the genetic programming system. In our work every one of the 100 fitness cases is iterated through in each individual evaluation. There are good reasons to believe that this number could be varied using just a few randomly picked samples from a large training set in every individual evaluation. Genetic algorithms have often been proved to perform well with an overall faster convergence time, in a noisy fitness environment, using only a subset of the fitness cases in each evaluation, for instance in image processing (Fitzpatrick *et al* 1984).

The memory consumption of a CGPS individual consisting of 12 instructions and 32 bits per instruction is about 50 bytes for a complete classifying program. The neural network needs 450 bytes if it is written in assembler on a computer with a floating point processor. On a computer without a floating point processor a stand alone implementation of the classifying neural network requires tens of thousands of bytes for linked libraries and the like.

The classification time is less than 1 microsecond with our hardware and the CGPS. The network could approach 500 μs in assembler on a floating point processor. On a computer without a floating point coprocessor it runs many times slower. The ratio is thus at least 500 times faster classification time. On a faster modern workstation, it is possible to run the CGPS individual 30 times faster, matching performance only otherwise achieved by hardware solutions.

Table G1.5.3 shows a summary of the comparison between the compiling genetic programming system and the neural network.

Table G1.5.3. Summary of comparison between a neural network and the CGPS.

	Neural network	CGPS
Training time (min)	235	1
Average % of examples learned in training set	89	86
Average % of examples correctly classified in unseen example set	69	72
Number of processed examples	4000	4 000 000
Storage consumption (bytes)	450–20 000	50
Classification time (μs)	500	1

G1.5.6 Applicability

There are many different possible areas of application for the CGPS. The ability to evolve small subroutines in machine code could be of use where there is a need for fast execution and/or small size. Examples of environments with limited performance and memory capability could be hand-held devices in consumer electronics running on one-chip processors. The small and fast training algorithm gives the possibility of 'on-board' training in most environments.

There are also applications with tight execution time limits, for instance fast real-time system control. The ability to evolve fast but complex solutions in 'fuzzy' problem spaces makes pattern recognition a potential application area, as well.

G1.5.7 Summary

In this article we have described a method to implement a compiling genetic programming system that evolves and manipulates the lowest-level binary machine code. The CGPS is up to 2000 times faster than a genetic programming system written in an interpreting language. The CGPS can be used for heuristic classification, and competes with the more established paradigm, neural networks. The learning and generalization capabilities are similar while the training time and classification time are more than two orders of magnitude faster with the CGPS in our example. This performance, combined with very low memory consumption, might make the system suitable for real-time applications in low-speed hardware or in applications where fast execution is required, such as hand-held devices, real-time pattern recognition, and process control.

Acknowledgement

This contribution includes some material previously published by the author in Nordin (1994), reproduced here by permission of The MIT Press.

References

Fitzpatrick J M, Grefenstette J J and Gucht D V 1984 Image registration by genetic search *Proc. IEEE Southeast Conf.* pp 460–4

Golberg D E, Korb B and Deb K 1989 Messy genetic algorithms: motivation analysis and first results *Complex Syst.* **3** 493–530

Koza J R 1992 *Genetic Programming* (Cambridge, MA: MIT Press)

Keith M J and Martin C 1993 Genetic programming in C++: implementation and design issues *Advances in Genetic Programming* ed K Kinnear Jr (Cambridge, MA: MIT Press)

Nordin J P 1994 A compiling genetic programming system that directly manipulates the machine-code *Advances in Genetic Programming* ed K Kinnear Jr (Cambridge, MA: MIT Press)

Nordin J P and Banzhaf W 1995 Evolving Turing complete programs for a register machine with self-modifying code *Proc. 6th Int. Conf. on Genetic Algorithms* (San Mateo, CA: Morgan Kaufmann)

Reynolds C W 1992 An evolved, vision-based behavioral model of coordinated group motion *From Animals to Animats 2: Proc. 2nd Int. Conf. on Simulation of Adaptive Behavior* ed J A Meyer (Cambridge, MA: MIT Press)

Rumelhart D E, Hinton G E and Williams R J 1986 Learning internal representation by error propagation *Parallel Distributed Processing* vol 1 (Cambridge, MA: MIT Press) pp 318–62

Sun Microsystems 1990 *Sun-4 Assembly Language Reference Manual* Part No 800-3806-10

G1.6 Evolving cellular automata to perform computations

Melanie Mitchell, James P Crutchfield and Rajarshi Das

Abstract

We describe an application of genetic algorithms (GAs) to the design of cellular automata (CAs) that can perform computations requiring global coordination. A GA was used to evolve CAs for two computational tasks: density classification and synchronization. In both cases, the GA discovered rules that gave rise to sophisticated emergent computational strategies. These strategies can be analyzed using a *computational mechanics* framework in which *particles* carry information and interaction between particles effects information processing. This framework can also be used to explain the process by which the strategies are designed by the GA. This work is a first step in employing GAs to engineer useful emergent computation in decentralized multiprocessor systems.

G1.6.1 Project overview

An example of automatic programming by *genetic algorithms* (GAs) is found in our work on evolving cellular automata (CAs) to perform computations (Mitchell *et al* 1993, Mitchell *et al* 1994, Crutchfield and Mitchell 1995, Das *et al* 1994, 1995; these papers can be obtained on the World Wide Web URL http://www.santafe.edu/projects/evca). This project has elements of both engineering and scientific modeling. One motivation is to understand how natural evolution creates systems in which *emergent computation* takes place—that is, in which the actions of simple components with local information and communication give rise to coordinated global information processing. Insect colonies, economic systems, the immune system, and the brain have all been cited as examples of systems in which such emergent computation occurs (Forrest 1990, Langton 1992). However, it is not well understood *how* these natural systems perform computations, let alone how evolution has produced them. Another motivation is to find ways to engineer sophisticated emergent computation in decentralized multiprocessor systems.

B1.2

One of the simplest decentralized, spatially extended systems in which emergent computation can be studied is a one-dimensional binary-state CA—a one-dimensional lattice of N two-state machines (*cells*), each of which changes its state as a function only of the current states in a local neighborhood. (The well-known Game of Life (Berlekamp *et al* 1982) is an example of a two-dimensional CA.) As illustrated in figure G1.6.1, the lattice starts out with an initial configuration (IC) of cell states (zeros and ones) and this configuration changes in discrete time steps in which all cells are updated simultaneously according to the CA *rule* ϕ. (Here we use the term *state* to refer to the value of a single cell. The term *configuration* will refer to the collection of local states over the entire lattice.)

A CA's rule ϕ can be expressed as a lookup table (*rule table*) that lists, for each local neighborhood, the state which is taken on by the neighborhood's central cell at the next time step. For a binary-state CA, these update states are referred to as the *output bits* of the rule table. In a one-dimensional CA, a neighborhood consists of a cell and its r (*radius*) neighbors on both sides. (In figure G1.6.1, $r = 1$.) Here we consider CAs with periodic boundary conditions—the lattice is viewed as a circle.

Cellular automata have been studied extensively as mathematical objects, as models of natural systems, and as architectures for fast, reliable parallel computation. (For overviews of CA theory and applications,

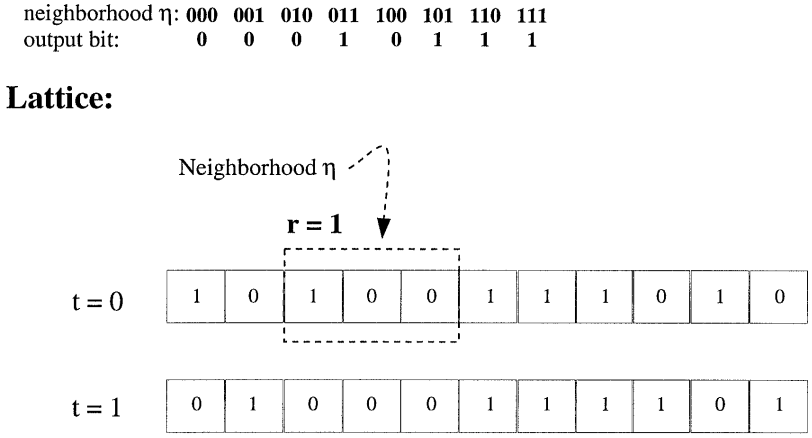


Figure G1.6.1. An illustration of a one-dimensional, binary-state, nearest-neighbor ($r = 1$) CA with $N = 11$. Both the lattice and the rule table ϕ for updating the lattice are illustrated. The lattice configuration is shown over one time step. The CA has spatially periodic boundary conditions: the lattice is viewed as a circle, with the leftmost cell being the right neighbor of the rightmost cell, and vice versa.

see Wolfram 1986 and Toffoli and Margolus 1987.) However, the difficulty of understanding the emergent behavior of CAs or of designing CAs to have desired behavior has up to now severely limited their use in science and engineering and for general computation. Here we describe work on using GAs to engineer CAs to perform computations.

Typically, a CA performing a computation means that the input to the computation is encoded as the IC, the output is decoded from the configuration reached at some later time step, and the intermediate steps that transform the input to the output are interpreted as the steps in the computation. The computation emerges from the CA rule being obeyed by each cell. (Note that this use of CAs as computers differs from the impractical, though theoretically interesting, method of constructing a universal Turing machine in a CA; see Mitchell *et al* 1993 for a comparison of these two approaches.)

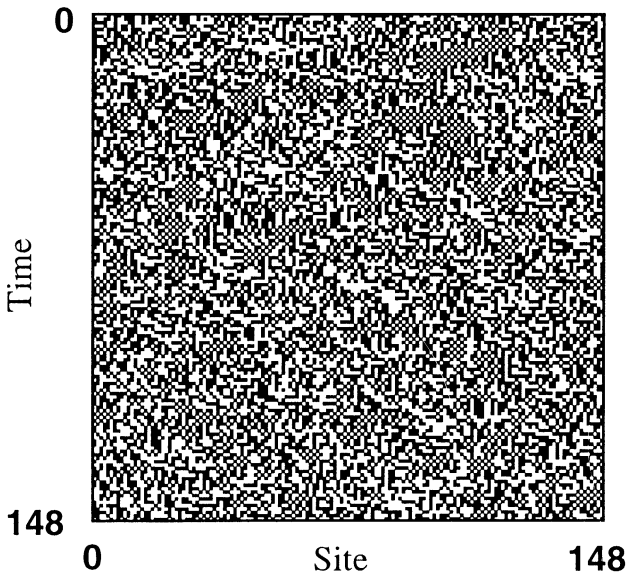


Figure G1.6.2. A space–time diagram for an $r = 3$ binary-state CA with an arbitrary rule table, iterating on a randomly generated initial configuration. $N = 149$ sites are shown, with time increasing down the page. Here cells with state zero are white and cells with state one are black. (This and the other space–time diagrams given here were generated using the program la1d written by James P Crutchfield.)

The behavior of one-dimensional binary-state CAs is often illustrated by a *space–time diagram*—a plot of lattice configurations over a range of time steps, with ones given as black cells and zeros given as white cells and with time increasing down the page. Figure G1.6.2 shows such a diagram for a binary-state $r = 3$ CA in which the rule table's output bits were filled in at random. The CA is shown iterating on a randomly generated IC. Apparently structureless configurations, such as those shown in figure G1.6.2, are typical for the vast majority of CAs. To produce CAs that can perform sophisticated parallel computations, the GA must search for CAs in which the actions of the cells, taken together, are coordinated so as to produce the desired behavior. This coordination must, of course, happen in the absence of any central processor or central memory directing the coordination.

Some early work on evolving CAs with GAs was done by Packard and colleagues (Packard 1988, Richards *et al* 1990). Koza (1992) also applied *genetic programming* to evolve CAs for simple random-number generation. B1.5.1

Our work builds on that of Packard (1988). We have used a form of the GA to evolve one-dimensional, binary-state $r = 3$ CAs to perform a density classification task (Crutchfield and Mitchell 1995, Das *et al* 1994) and a synchronization task (Das *et al* 1995).

G1.6.2 Design process

For the density classification task, the goal was to find a CA that decides whether or not the IC contains a majority of ones (i.e. has high density). If it does, the whole lattice should eventually produce an unchanging configuration of all ones; otherwise it should eventually go to all zeros. More formally, we call this task the $\rho_c = 1/2$ task. Here ρ denotes the density of ones in a binary-state CA configuration and ρ_c denotes a *critical* or threshold density for classification. Let ρ_0 denote the density of ones in the IC. If $\rho_0 > \rho_c$, then within M time steps the CA should go to the fixed-point configuration of all ones (i.e. all cells in state one for all subsequent iterations); otherwise, within M time steps it should produce the fixed-point configuration of all zeros. M is a parameter of the task that depends on the lattice size N.

Designing an algorithm to perform the $\rho_c = 1/2$ task is trivial for a system with a central controller or central storage of some kind, such as a finite-state machine with a counter register or a neural network in which all input units are connected to a central hidden unit. However, the task is nontrivial for a small-radius ($r \ll N$) CA, since a small-radius CA relies only on local interactions. It has been argued that no finite-radius, binary-state CA with periodic boundary conditions can perform this task perfectly across all lattice sizes (Land and Belew 1995, Das 1996), but even to perform this task well for a fixed lattice size requires more powerful computation than can be performed by a single cell or any linear combination of cells. Since the ones can be distributed throughout the CA lattice, the CA must transfer information over large distances ($\approx N$). To do this requires the global coordination of cells that are separated by large distances and that cannot communicate directly. How can this be done? Our interest was to see whether the GA could devise one or more methods.

The chromosomes evolved by the GA were *bit strings* representing CA rule tables with $r = 3$. Each chromosome consisted of the output bits of a rule table, listed in lexicographic order of neighborhood (cf ϕ in figure G1.6.1). The chromosomes representing rules were thus of length $2^{2r+1} = 128$. The size of the rule space in which the GA worked was 2^{128}—far too large for any kind of exhaustive evaluation. C1.2

In our main set of experiments, we set $N = 149$, a reasonably large but still computationally tractable odd number (odd, so that the task will be well defined on all ICs). The GA began with a population of 100 randomly generated chromosomes (generated with some initial biases—see Mitchell *et al* 1994 for details). The fitness of a rule in the population was computed by (i) randomly choosing 100 ICs that are uniformly distributed over $\rho \in [0.0, 1.0]$, with exactly half with $\rho < \rho_c$ and half with $\rho > \rho_c$, (ii) iterating the CA on each IC either until it arrives at a fixed point or for a maximum of $M \approx 2N$ time steps, and (iii) determining whether the final behavior is correct—that is, 149 zeros for $\rho_0 < \rho_c$ and 149 ones for $\rho_0 > \rho_c$. The initial density, ρ_0, was never exactly $1/2$, since N was chosen to be odd. The rule's fitness, F_{100}, was the fraction of the 100 ICs on which the rule produced the correct final behavior. No partial credit was given for partially correct final configurations.

A few comments about the fitness function are in order. First, the number of possible input cases (2^{149} for $N = 149$) was far too large for fitness to be defined as the fraction of correct classifications over all possible ICs. Instead, fitness was defined as the fraction of correct classifications over a sample of 100 ICs. A different sample was chosen at each generation, making the fitness function stochastic. In addition, the ICs were not sampled from an unbiased distribution (i.e. equal probability of a one or a zero at each

site in the IC), but rather from a flat distribution across $\rho \in [0, 1]$ (i.e. ICs of each density from $\rho = 0$ to $\rho = 1$ were approximately equally represented). This flat distribution was used because the unbiased distribution is binomially distributed and thus very strongly peaked at $\rho = 1/2$. The ICs selected from such a distribution will likely all have $\rho \approx 1/2$, the hardest cases to classify. Using an unbiased sample made it more difficult for the GA to discover high-fitness CAs.

Our version of the GA worked as follows. In each generation, (i) a new set of 100 ICs was generated, (ii) F_{100} was computed for each rule in the population, (iii) CAs in the population were ranked in order of fitness, (iv) the 20 highest-fitness (*elite*) rules were copied to the next generation without modification, and (v) the remaining 80 rules for the next generation were formed by single-point crossovers between randomly chosen pairs of elite rules. The parent rules were chosen from the elite with replacement—that is, an elite rule was permitted to be chosen any number of times. The offspring from each crossover were each mutated at exactly two randomly chosen positions. This process was repeated for 100 generations for a single run of the GA. (More details of the implementation are given by Mitchell *et al* 1994.)

Our selection scheme, in which the top 20% of the rules in the population are copied without modification to the next generation and the bottom 80% are replaced, is similar to the $\mu + \lambda$ selection method used in some *evolution strategies* (see Bäck *et al* 1991). Selecting parents by relative fitness rank B1.3 rather than in proportion to absolute fitness helps to prevent initially stronger individuals from too quickly dominating the population and driving the genetic diversity down too early. Also, since testing a rule on 100 ICs provides only an approximate gauge of the rule's performance over all 2^{149} possible ICs, saving the top 20% of the rules was a good way of making a first cut and allowing rules that survive to be tested over different ICs. Since a new set of ICs was produced every generation, rules that were copied without modification were always retested on this new set. If a rule performed well and thus survived over a large number of generations, then it was likely to be a genuinely better rule than those that were not selected, since it was tested with a large set of ICs.

G1.6.3 Results

Three hundred different runs were performed, each starting with a different random-number seed. On most runs the GA evolved a rather unsophisticated class of strategies. One example, a CA here called ϕ_a, is illustrated in the top part of figure G1.6.3. This rule had $F_{100} \approx 0.9$ in the generation in which it was discovered. Its computational strategy is the following: Quickly reach the fixed point of all zeros unless there is a sufficiently large block of adjacent (or almost adjacent) ones in the IC. If so, expand that block. (For this rule, *sufficiently large* is seven or more cells.) This strategy does a fairly good job of classifying low- and high-density ICs under F_{100}: it relies on the appearance or absence of blocks of ones to be good predictors of ρ_0, since high-density ICs are statistically more likely to have blocks of adjacent ones than low-density ICs.

Similar strategies were evolved in most runs. On approximately half the runs, *expand ones* strategies were evolved, and on most of the other runs, the opposite *expand zeros* strategies were evolved. These block-expanding strategies do not count as sophisticated examples of emergent computation in CAs: all the computation is done locally in identifying and then expanding a sufficiently large block. There is no global coordination or sophisticated information flow between distant cells—two things we claimed were necessary to perform well on the task. Indeed, such strategies perform poorly under performance measures using different (e.g. unbiased) distributions of ICs, and when N is increased.

Mitchell *et al* (1994) analyzed the detailed mechanisms by which the GA evolved such block-expanding strategies. This analysis uncovered several notable aspects of the GA, including a number of impediments that, on most runs, kept the GA from discovering better-performing CAs. These included the GA breaking the $\rho_c = 1/2$ task's symmetries for short-term gains in fitness, as well as overfitting to the fixed lattice size $N = 149$ and the unchallenging nature of the IC samples at late generations. These impediments are discussed in detail by Mitchell *et al* (1994), but the last point merits some elaboration here.

The biased, flat distribution of ICs over $\rho \in [0, 1]$ helped the GA to move away from zero fitness in the early generations. We found that computing fitness using an unbiased distribution of ICs made the problem too difficult for the GA early on—it was rarely able to find improvements to the CAs in the initial population. However, the biased distribution became too easy for the improved CAs later in a run, and these ICs did not push the GA hard enough to find better solutions. We are currently exploring a coevolution scheme to improve the GA's performance on this problem.

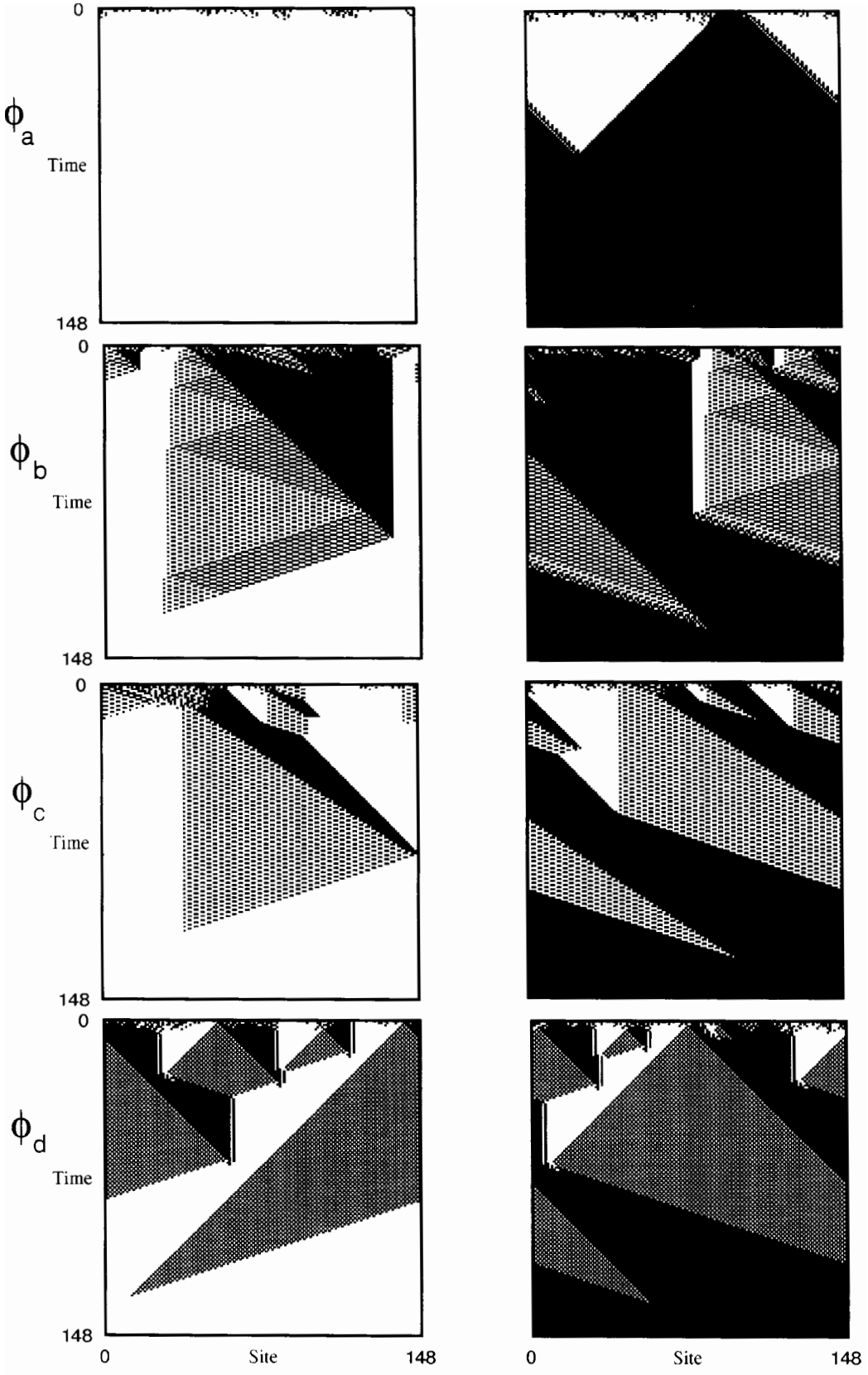


Figure G1.6.3. Space–time diagrams from four different rules discovered by the GA. The left diagrams have $\rho_0 < 1/2$; the right diagrams have $\rho_0 > 1/2$. All are correctly classified. Fitness increases from top to bottom. (Reprinted from Das *et al* 1994 with permission of the publisher; copyright 1994 Springer-Verlag.)

Despite these various impediments and the unsophisticated CAs evolved on most runs, on approximately 2% of the runs in our initial experiment the GA discovered CAs with more sophisticated strategies that yielded significantly better performance across different IC distributions and lattice sizes than was achieved by block-expanding strategies. The typical space–time behaviors of three such rules, ϕ_a, ϕ_c, and ϕ_d (each from a different run), are illustrated in figure G1.6.3.

For example, ϕ_d was the best-performing rule discovered in our initial GA experiments. In the bottom part of figure G1.6.3 it can be seen that, under ϕ_d, there is a transient phase during which spatial and temporal transfer of information about the density in local regions takes place. This local information interacts with other local information to produce the desired final state. Roughly, ϕ_d successively classifies local densities with a locality range that increases with time. In regions where there is some ambiguity, a 'signal' is propagated. This is seen either as a checkerboard pattern propagated in both spatial directions or as a vertical black-to-white boundary. These signals indicate that the classification is to be made later at a larger scale. The creation and interactions of these signals can be interpreted as the locus of the computation being performed by the CA—they form its emergent program.

The above explanation of how ϕ_d performs the $\rho_c = 1/2$ task is an informal one obtained by careful scrutiny of many space–time diagrams. Can we understand more rigorously how the evolved CAs perform the desired computation? Understanding the results of GA evolution is a general problem—typically the GA is asked to find individuals that achieve high fitness but is not told what traits the individuals should have to attain high fitness. One could say that this is analogous to the difficulty biologists have in understanding the products of natural evolution. We computational evolutionists have similar problems, since we do not specify *what* solution evolution is supposed to create; we ask only that it find *some* good solution. In many cases, it is difficult to understand exactly how an evolved high-fitness individual works. The problem is especially difficult in the case of CAs, since the emergent computation performed by a given CA is almost always impossible to deduce from the bits of the rule table.

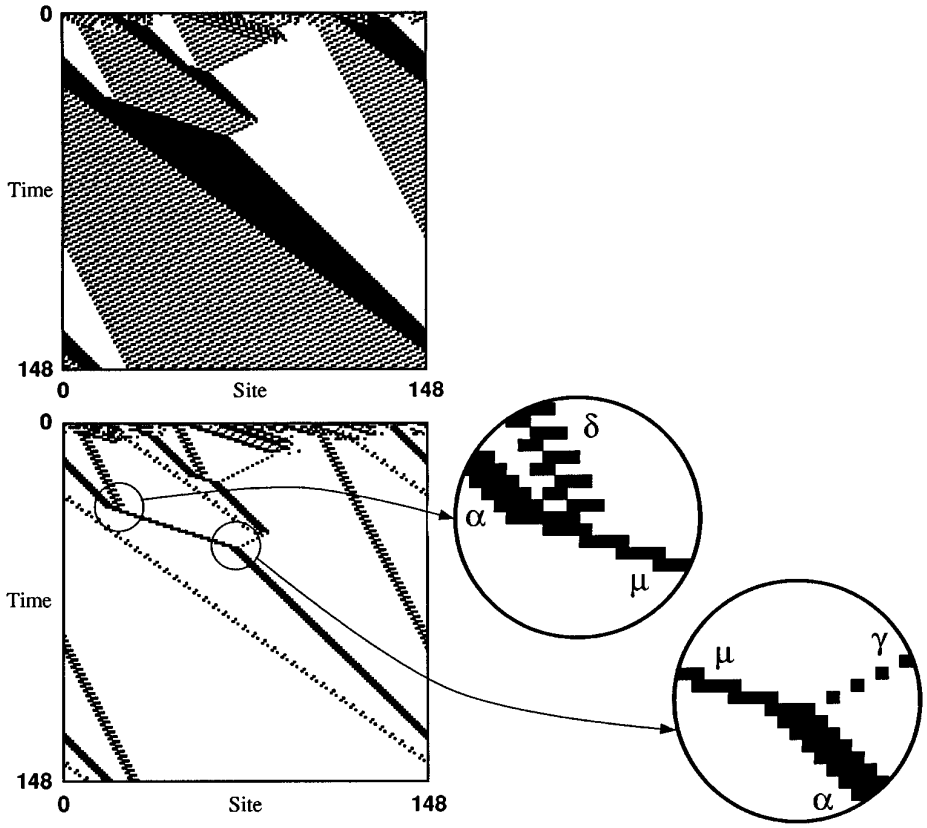


Figure G1.6.4. A space–time diagram of a GA-evolved rule for the $\rho_c = 1/2$ task, and the same diagram with the regular domains filtered out, leaving only the particles and particle interactions (two of which are magnified). (Reprinted from Crutchfield and Mitchell 1995 by permission of the authors.)

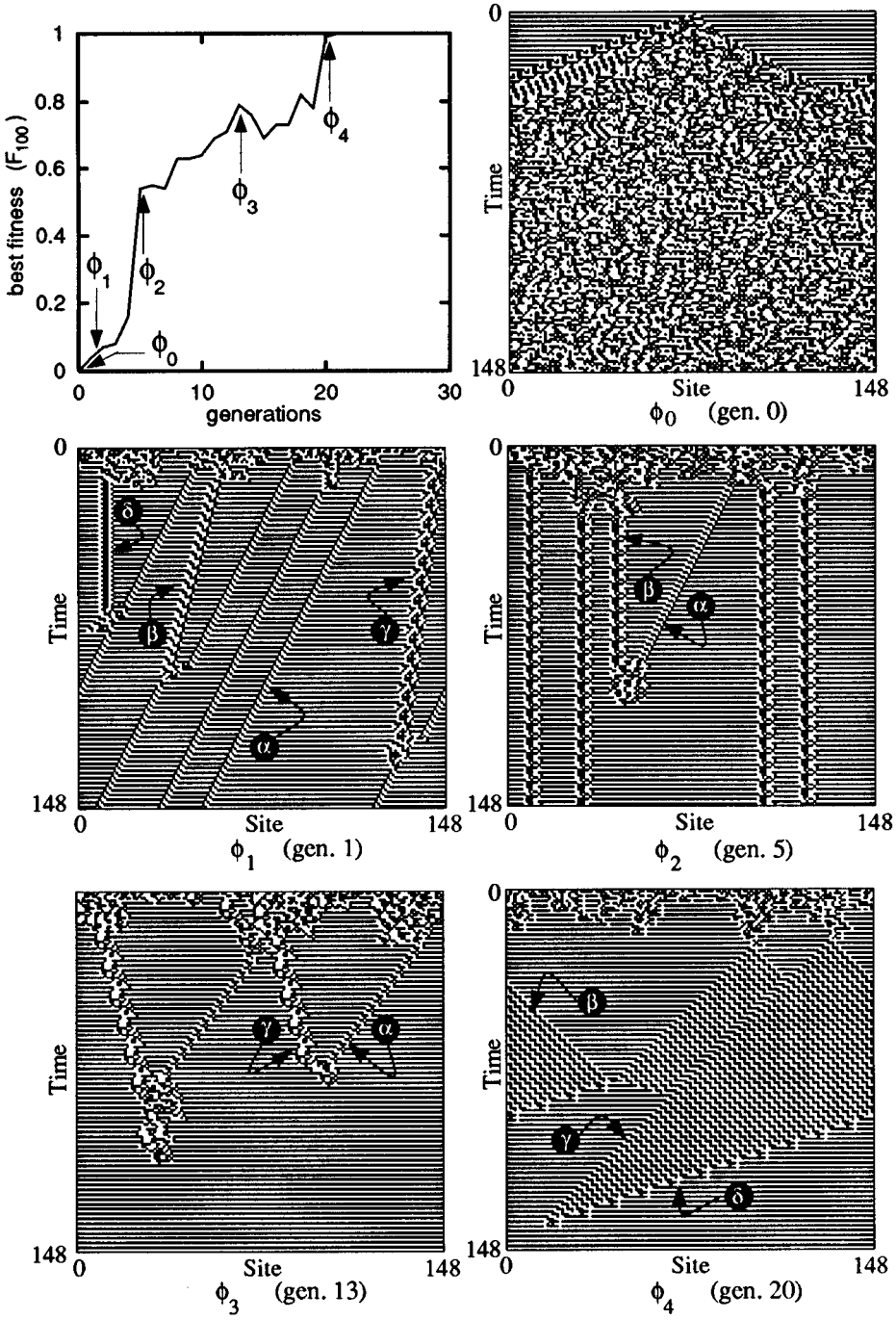


Figure G1.6.5. The evolutionary history of one GA run on the synchronization task: top left: F_{100} versus generation for the most fit CA in each population. The arrows indicate the generations in which the GA discovered each new significantly improved strategy. Other plots: space–time diagrams illustrating the behavior of the best ϕ at each of the five generations marked in the top left. The ICs are random except for the top right, which consists of a single one in the center of a field of zeros. The same Greek letters in different figures represent different types of particle. (Reprinted from Das *et al* 1995 by permission of the authors.)

Instead, our approach is to examine the space–time behavior exhibited by the CA and to *reconstruct* from that behavior what the emergent algorithm is. Crutchfield and Hanson have developed a general method for reconstructing and understanding the 'intrinsic' computation embedded in space–time behavior in terms of 'regular domains', 'particles', and 'particle interactions' (Hanson and Crutchfield 1992,

Crutchfield and Hanson 1993). This method is part of their 'computational mechanics' framework for understanding computation in physical systems (Crutchfield 1994). A detailed discussion of computational mechanics and particle-based computation is beyond the scope of this article. Very briefly, for those familiar with formal language theory, regular domains are regions of space–time consisting of words in the same regular language—in other words, they are regions that are computationally homogeneous and simple to describe. Particles are the localized boundaries between those domains. In computational mechanics, particles can be shown to be information carriers, and collisions between particles are identified as the loci of information processing. Particles and particle interactions form a high-level language for describing computation in spatially extended systems such as CAs. Figure G1.6.4 hints at this higher level of description: to produce it we filtered the regular domains from the space–time behavior of a GA-evolved CA to leave only the particles and their interactions, in terms of which the emergent algorithm of the CA can be understood.

The application of computational mechanics to analyzing the rules evolved by the GA is discussed further by Das *et al* (1994, 1995), and by Crutchfield and Mitchell (1995). In the first two papers, we used particles and particle interactions to describe the evolutionary epochs by which highly fit rules were evolved by the GA. An illustration of the succession of these epochs for the synchronization task is given in figure G1.6.5. The goal for the GA was to find a CA that, from any IC, produces a globally synchronous oscillation between the all-ones and all-zeros configurations. (This is perhaps the simplest version of the emergence of spontaneous synchronization that occurs in decentralized systems throughout nature.) The computational-mechanics analysis allowed us to understand the evolutionary innovations produced by the GA in the higher-level language of particles and particle interactions as opposed to the low-level language of CA rule tables and spatial configurations.

G1.6.4 Conclusions

The discoveries of rules such as ϕ_b–ϕ_d and of rules that produce global synchronization is significant, since these are the first examples of a GA producing sophisticated emergent computation in decentralized, distributed systems such as CAs. These discoveries made by a GA are encouraging for the prospect of using GAs to automatically evolve computation for more complex tasks (e.g. image processing or image compression) and in more complex systems. Moreover, evolving CAs with GAs also gives us a tractable framework in which to study the mechanisms by which an evolutionary process might create complex coordinated behavior in natural decentralized and distributed systems. For example, by studying the GA's behavior, we have already learned how evolution's breaking of symmetries can lead to suboptimal computational strategies (Mitchell *et al* 1993); eventually we may be able to use such *simulation models* F1.8 to test ways in which such symmetry breaking might occur in natural evolution. In general, models such as ours can provide insights into how evolutionary processes can discover structural properties of individuals that give rise to improved adaptation. In our case, such structural properties—regular domains and particles—were identified via the computational mechanics framework (Crutchfield 1994), and allowed us to analyze the evolutionary emergence of sophisticated computation.

Acknowledgments

This work was supported by the Santa Fe Institute under grants from the National Science Foundation (IRI-9320200), the Department of Energy (DE-FG03-94ER25231), and the Defense Advanced Projects Research Agency and the Office of Naval Research (N00014-95-1-0975). It was supported by the University of California, Berkeley, under grants from the Office of Naval Research (N00014-95-1-1524) and the Air Force Office of Scientific Research (91-0293).

References

Bäck T, Hoffmeister F and Schwefel H-P 1991 A survey of evolution strategies *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, July 1991)* ed R Belew and L Booker (San Mateo, CA: Morgan Kaufmann) pp 2–9

Berlekamp E, Conway J H and Guy R 1982 *Winning Ways for Your Mathematical Plays* vol 2 (New York: Academic)

Crutchfield J P 1994 The calculi of emergence: computation, dynamics, and induction *Physica* **75D** 11–54

Crutchfield J P and Hanson J E 1993 Turbulent pattern bases for cellular automata *Physica* **69D** 279–301

Crutchfield J P and Mitchell M 1995 The evolution of emergent computation *Proc. Natl Acad. Sci., USA* **92** 10 742

Das R 1996 *The Evolution of Emergent Computation in Cellular Automata* PhD Thesis, Computer Science Department, Colorado State University

Das R, Crutchfield J P, Mitchell M and Hanson J E 1995 Evolving globally synchronized cellular automata *Proc. 6th Int. Conf. on Genetic Algorithms (Pittsburgh, PA, July 1995)* ed L J Eshelman (San Mateo, CA: Morgan Kaufmann) pp 336–43

Das R, Mitchell M and Crutchfield J P 1994 A genetic algorithm discovers particle-based computation in cellular automata *Parallel Problem solving from Nature—PPSN III (Proc. Int. Conf. on Evolutionary Computation and 3rd Conf. on Parallel Problem Solving from Nature, Jerusalem, October 1994)* (*Lecture Notes in Computer Science 866*) ed Yu Davidor, H-P Schwefel and R Männer (Berlin: Springer) pp 244–353

Forrest S 1990 Emergent computation: self-organizing, collective, and cooperative phenomena in natural and artificial computing networks *Physica* **42D** 1–11

Hanson J E and Crutchfield J P 1992 The attractor-basin portrait of a cellular automaton *J. Stat. Phys.* **66** 1415–62

Koza J R 1992 *Genetic Programming: On the Programming of Computers by Means of Natural Selection* (Cambridge, MA: MIT Press)

Land M and Belew R K 1995 No perfect two-state cellular automata for density classification exists *Phys. Rev. Lett.* **74** 5148

Langton C G 1992 Introduction *Artificial Life II* ed C G Langton, C Taylor, J D Farmer and S Rasmussen (Reading, MA: Addison-Wesley) pp 3–23

Mitchell M, Crutchfield J P and Hraber P T 1994 Evolving cellular automata to perform computations: mechanisms and impediments *Physica* **75D** 361–91

Mitchell M, Hraber P T and Crutchfield J P 1993 Revisiting the edge of chaos: evolving cellular automata to perform computations *Complex Syst.* **7** 89–130

Packard N H 1988 Adaptation toward the edge of chaos *Dynamic Patterns in Complex Systems* ed J A S Kelso, A J Mandell, M F Shlesinger (Singapore: World Scientific) pp 293–301

Richards F C, Meyer T P and Packard N H 1990 Extracting cellular automaton rules directly from experimental data *Physica* **45D** 189–202

Toffoli T and Margolus N 1987 *Cellular Automata Machines: A New Environment for Modeling* (Cambridge, MA: MIT Press)

Wolfram S 1986 *Theory and Applications of Cellular Automata* (Singapore: World Scientific)

G1.7 Application of a genetic algorithm to finding high-performance configurations for surface mount device assembly lines

J David Schaffer

Abstract

In 1993 Philips Industrial Electronics Corp. brought to market a new surface mount device assembly robot with significantly higher performance than the existing state of the art. A key component of this product was an optimizer that used a genetic algorithm (GA) to find high-performance setups for a production line of these robots for any given assembly task including the possibility of user-imposed constraints. The development of this optimizer is described with emphasis on the key features, particularly the heuristic setup generator and the robust GA employed. For the finer details the reader is referred to a US patent.

G1.7.1 Project overview

In the early 1990s Philips Industrial Electronics launched a development project to produce a multiheaded robot for placing surface mount devices (SMDs—chips) on printed circuit boards (PCBs). The goal was a modular machine design that would significantly increase the industry high-speed standard of 30 000 components placed per hour. The fast component mounter (FCM) concept called for up to 16 independent heads in a row working above a transport system carrying the PCBs. Philips knew that customers for SMD assembly equipment would not purchase such machines unless accompanied by software to solve the 'setup problem'. An algorithm had to be provided that, given a description of a PCB (a list of components with their placement coordinates) and a description of a production line (the number of FCMs with the number of heads on each and any preassigned tooling), would return a setup description with which the PCB can be manufactured in minimum time (cycle time). A setup consists of an assignment of chip-handling tooling to each robot head, an assignment of chip feeders to the available feeder bars, and a list of specific chips to be placed by each robot head during each index step of the transport system. This is a discrete nonlinear combinatorial problem that can be of immense size, daunting complexity (containing several NP-complete subtasks), and with nonlinear constraints. In early 1993 the FCM machine was demonstrated at an industry trade show placing over 60 000 components per hour using setups found automatically by a *genetic algorithm* (GA) optimizer. Today the FCM is in use by major electronics manufacturers all over the world. A sketch of the FCM is shown in figure G1.7.1. B1.2

The core problem was solved in concurrent engineering mode (the optimizer was developed in parallel with the FCM machine itself) in a year by a team of three: a software engineer at Philips I&E and two GA researchers at the corporate laboratory. The approach taken was to develop a heuristic setup generator (HSG) coupled to a GA as shown schematically in figure G1.7.2. This approach had proven more successful than an alternative based on simulated annealing in exploratory trials on an older machine design. The inherent robustness of the GA was important in the concurrent engineering mode where optimizer code often had to be written with 'best-guess' models before essential machine design decisions had been finalized.

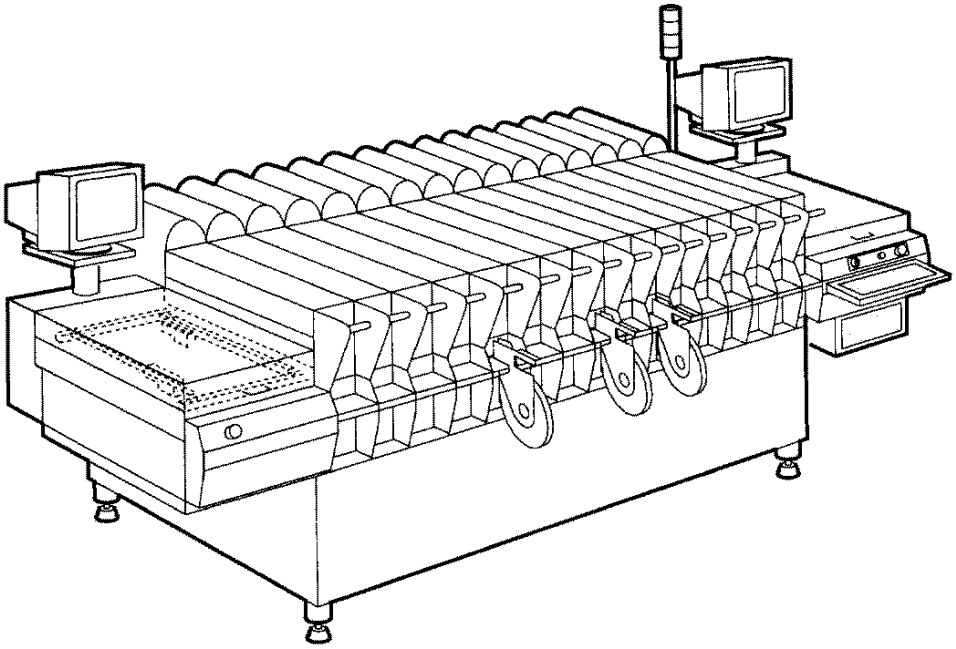


Figure G1.7.1. A sketch of the FCM SMD assembly machine.

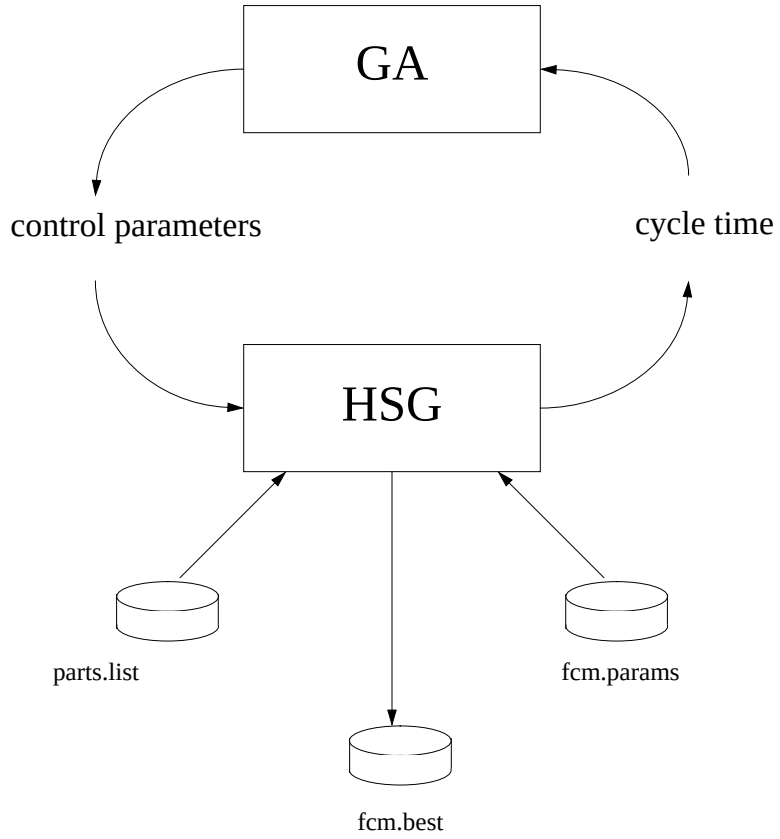


Figure G1.7.2. The GA linked to the HSG.

G1.7.2 The approach

Since a number of the subtasks involved in generating a complete layout (assigning tooling to robot heads, assigning feeders to feeder bars, and assigning parts to heads at index steps) are well-studied problem types (e.g. *set covering*, *bin packing*, and line balancing) with significant accumulated knowledge of heuristics, it was tempting to try to divide and conquer: attack each subtask as a separate problem using the best known heuristics. However, this approach had been tried by others within Philips using *simulated annealing* and *local search* methods and it had not worked well. The main reason is the lack of good measures of subtask quality (e.g. it is hard to know how good an assignment of tooling to heads is without knowing the rest of the setup). However, with a complete setup in hand, assessing the quality of the ensemble is easy; the cycle time of the robot can be accurately modeled and calculated.

G9.6, F1.7

D3.5

D3.2

By developing a heuristic setup generator, we could employ a number of well-studied heuristics from the literature and combine them with known properties of the FCM (domain knowledge). However, we deemed it beyond our skill to combine this knowledge into a single-pass optimization algorithm capable of good performance for the whole range of PCB manufacturing challenges. To add the needed flexibility, we embedded in the HSG many parameters capable of modifying the behavior of the algorithm. The chromosome consisted of a string of these parameters and the task for the GA was to tune the set of parameters so that the HSG generated good setups for any given PCB.

We were optimistic that this could be done because of our own earlier experiments and because a similar approach had been used by others to good effect (e.g. Kadaba and Nygard 1990, Syswerda and Palmucci 1991). Furthermore, our experience with Eshelman's *CHC algorithm* (Eshelman 1991) had shown us its robustness and efficiency over a wide range of problem types.

B1.2.2

The general structure of the HSG is illustrated in figure G1.7.3. The chromosome is coded as a bitstring consisting of several Gray-coded numeric parameters and a large number of binary decision variables. As shown in figure G1.7.3, some genes influence the decisions made in each of the subtasks, but only when a complete setup has been produced (analogous to growing a phenotype from a genotype) can we assess its performance by running the timing model and calculating the cycle time. The challenge to the GA is to find an ensemble of genes that work together to yield high-performance setups.

One of the ways we used the embedded parameters deals with the lack of good measures of subtask quality. We devised heuristic measures of 'desirability' for each subtask where we knew approximately, but not very precisely, what was desirable. We then also included in the desirability calculus weighting terms which were supplied by genes. This gave the GA an ability to seek the right proxy measures to use for each subtask for each PCB. In addition, other genes were capable of modifying the decisions made while pursuing the local proxy measure, yielding an algorithm capable of generating a wide variety of setups under gene control. The heuristics used in one subtask are described in the next section to give a flavor of the approach. The interested reader is referred to the US patent for more details (Eshelman and Schaffer 1995).

G1.7.3 The line balancing heuristics

To illustrate the HSG approach, we describe the heuristics with their modifying genes for the *assign parts* subtask in figure G1.7.3. The routine is called *level*. It starts with a production line with tooling assigned to each robot head and all feeder bars filled with empty feeders (unless some have been preassigned with specific chip types by the user). Its job is to assign each component on the PCB to be placed by a specific head in a specific index step of the transport system.

Figure G1.7.4 illustrates the main data structure used by *level*, called the bucket array. There is a bucket for each head for each index step. All chips placed by a given head must be compatible with the tooling on that head and must be fed by one of the feeders assigned to that head. In addition, to be a 'possible' bucket for a given chip, the PCB location for that chip must be in the area reachable by the head in that index step. The pick-and-place time required for each chip can be estimated as the round trip travel time from pickup to place position and back to the pickup location (to fetch the next chip). Since heads place one chip at a time, the execution time for all chips in a bucket is a simple sum. Since the transport system cannot take the next step forward until the slowest head in the line has placed its last chip (the heartbeat time of the index step), it is desirable to balance the workload over heads. The desirability measure used by *level* is the slack time in each bucket: the difference between the bucket's estimated pick-and-place time and the cycle time. Since the quality of the resulting assignments will be sensitive to

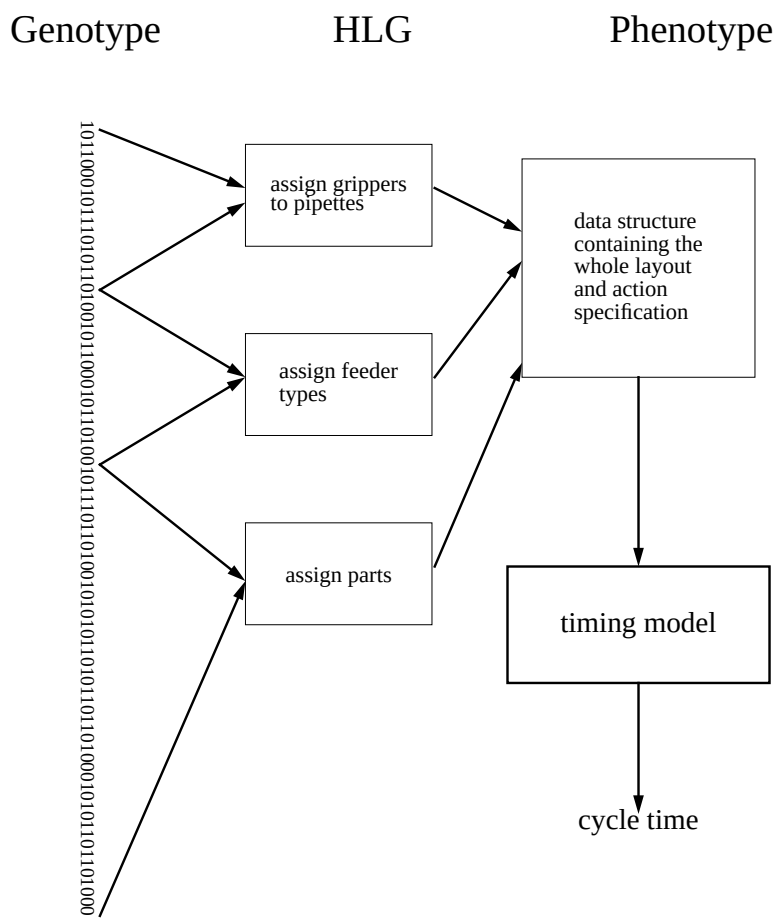


Figure G1.7.3. A schematic of the HSG.

the order in which the chips are assigned, the chromosome has genes that determine this sort order. In addition, as each chip is assigned to a head, a feeder must be committed to that chip type. This creates the possibility that the simple greedy *level* algorithm may 'paint itself into a corner' by giving away so many feeders to the first chips in the list that it finds itself with no possible buckets for the chips later in the list. To give the GA some control over this, there is a *bucket choice bit* for each chip. This bit is invoked when there is more than one possible bucket for a chip and one or more of them already possess a feeder committed to the required chip type. If the bucket choice bit is not set, the bucket with maximum slack time is selected. Otherwise, the choice is restricted to those buckets that would not require committing a new feeder, thus conserving free feeders for chips later in the list. These control genes give *level* an ability to behave as a complex mixture of best-first and best-first while conserving feeders algorithms.

G1.7.4 The genetic algorithm

The GA used in this product is Eshelman's CHC (Eshelman 1991) and is virtually identical to the algorithm used in our previous work (Schaffer and Eshelman 1993). It uses a population size of 50 for all PCBs and has demonstrated its robustness for problems with chromosomes ranging from 150 to over 3000 bits with no additional tuning of parameters. CHC's mechanisms include

- strictly cross-generational rank selection (the population always consists of the 50 best individuals ever generated)
- all population members having one opportunity to mate once each generation
- self-tuning incest prevention (Eshelman and Schaffer 1991)
- a vigorous form of crossover (half uniform crossover (Schaffer *et al* 1991) applied to all matings
- no mutation until population convergence, at which point a soft restart is performed.

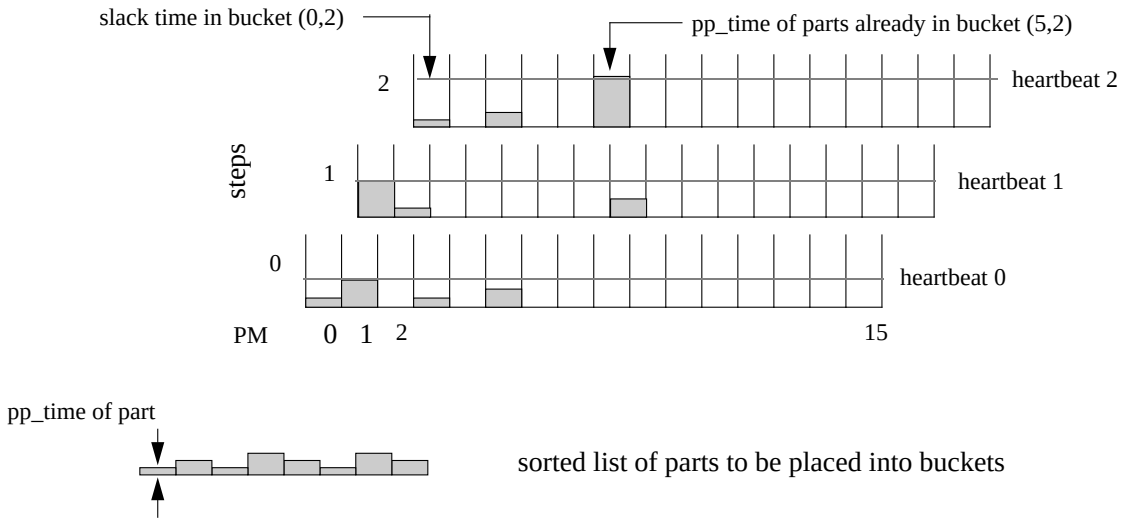


Figure G1.7.4. The bucket data structure filled by the *level* algorithm.

A soft restart involves making 50 copies of the best individual in the population, preserving one intact and applying a high rate of mutation (25% of bits are flipped) to the others. Then the normal selection and crossover mechanisms operate until the next convergence. The algorithm can be halted after a prespecified number of offspring have been generated, but we prefer to halt after a given number of restarts (ten for FCM tasks).

We have observed that problem difficulty does not correlate strictly with chromosome length (length is closely related to the number of chips on the PCB). Some cases are repeatedly solved to the same solution in fewer than 10 000 trials while others run for more than 100 000 with considerable variance among repeated runs with different random seeds, but run length and variance among results are not highly correlated either. Algorithm performance is discussed in the next section.

G1.7.5 Algorithm performance

Since we are unable to know the true optima for FCM problems, we investigated the algorithm's performance in three other ways: by comparing to lower bounds, by trying other search algorithms and by competing with human experts. While none of these methods is adequate to prove this approach cannot be improved (in fact we believe it can be), the evidence gathered was sufficient to decide to go to market with this approach.

We can produce a crude lower bound by simply summing the estimated pick-and-place times for all the chips on a PCB and dividing by the number of robot heads. The difference between this bound and the best feasible solution found by the algorithm is called the optimality gap. Experience has shown that optimality gaps vary from a few percent above the lower bound for some PCBs to more than twice the lower bound for others. However, the reasons for these wide differences are usually apparent. Some have to do with the distribution of the number of chips of each type. For example, if there is only one instance of a large chip on a PCB, then one head may have to be dedicated to placing that chip because it needs special tooling or a wide feeder. That one head may have to be idle much of the time, forcing a large optimality gap. Other PCBs may have large areas where no chips are placed (perhaps the space is reserved for special non-SMD components). When the PCB real estate with little or no work to do is in the working area of a robot head, it must experience idle time. Other PCBs have so many chip types that the challenge is to find any feasible feeder packing and no latitude is left for achieving good leveling. Our experience to date has been that there was always such an explanation when optimality gaps were large.

Another way to examine algorithm performance involves removing the GA and linking other search procedures to the HSG. Perhaps the simplest is a bit-flipping hillclimber (Davis 1991). Figure G1.7.5 shows a frequency distribution of local minima found by such a bit climber on one of our PCB test cases. One thousand independent climbs consumed over 500 000 trials and yielded two solutions with cycle times better than 10.2 seconds. In contrast, six runs of the GA consumed fewer than 420 000 trials and yielded

six solutions, all better than 10.2 seconds. This comparison is one of the better ones for the hillclimber whose performance on more difficult test cases was even more dramatically inferior to the GA. Similar experiments were performed with a simulated annealer whose state change operator also flipped single bits. Performance similar to the GA could be achieved, but only after much tuning of the cooling schedule and this tuning had to be repeated for each new PCB. Tuning the algorithm for each PCB could not be tolerated.

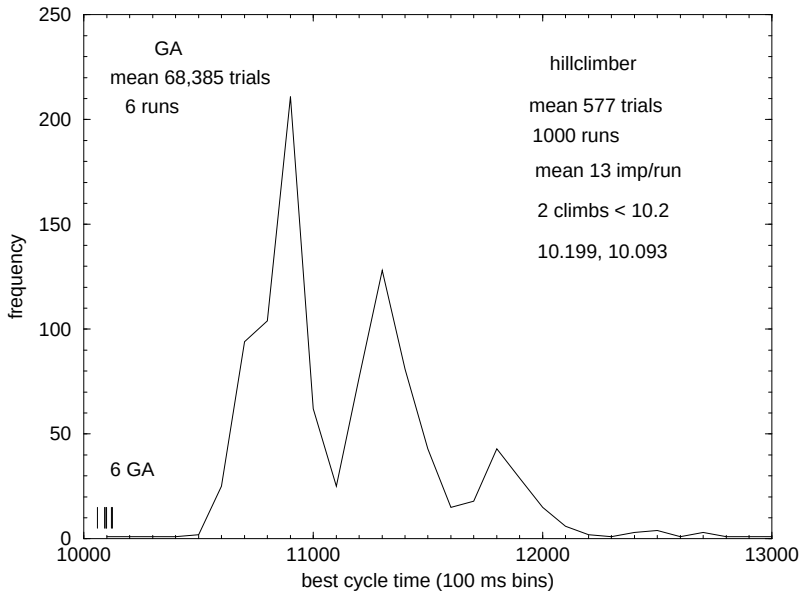


Figure G1.7.5. A frequency distribution of local minima for one PCB setup task.

Finally, some effort was made by human experts with modest computer assistance to improve upon the solutions reported by the GA. The experts tried small perturbations, say switching some feeders or reassigning some chips from some heads to others and recomputing cycle times. These efforts invariably failed and were soon abandoned.

G1.7.6 Conclusions

The GA solution to finding high-performance setups for production lines of Philips FCM machines has been a commercially successful product for over 3 years. Features worth noting about this approach include:

- The inherent robustness of the GA was important during the development stage because the demands of concurrent engineering (i.e. constantly changing details) precluded brittle approaches.
- The self-tuning nature of CHC was critical because the domain presents problems with quite different challenges (some present a challenge to find any feasible packing onto the machine while others are easily fitted on the machine and the challenge is good leveling) and tuning the algorithm for each new problem would not be tolerated.
- The approach of using a parameterized HSG allowed the straightforward inclusion of domain knowledge and good heuristics from the literature, as well as the use of the well-tested genetic operators.

We are optimistic that this approach will prove effective for a wide range of commercially important discrete optimization problems.

References

Davis L 1991 Bit-climbing, representational bias, and test suite design *Proc. 4th Int. Conf. on Genetic Algorithms* ed R K Belew and L B Booker (San Mateo, CA: Morgan Kaufmann) pp 18–23

Eshelman L E 1991 The CHC adaptive search algorithm: how to have safe search when engaging in nontraditional genetic recombination *Foundations of Genetic Algorithms* ed G J E Rawlins (San Mateo, CA: Morgan Kaufmann) pp 265–83

Eshelman L J and Schaffer J D 1991 Preventing premature convergence in genetic algorithms by preventing incest *Proc. 4th Int. Conf. on Genetic Algorithms* ed R K Belew and L B Booker (San Mateo, CA: Morgan Kaufmann) pp 115–22

——1995 *Method for Optimizing the Configuration of a Pick and Place Machine* United States Patent 5390283

Kadaba N and Nygard K E 1990 *Improving the Performance of Genetic Algorithms in Automated Discovery of Parameters* Department of SC and OR, North Dakota State University

Schaffer J D and Eshelman L J 1993 Designing multiplierless digital filters using genetic algorithms *Proc. 5th Int. Conf. on Genetic Algorithms* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 439–44

Schaffer J D, Eshelman L J and D Offutt 1991 Spurious correlations and premature convergence in genetic algorithms *Foundations of Genetic Algorithms* ed G J E Rawlins (San Mateo, CA: Morgan Kaufmann) pp 102–12

Syswerda G and Palmucci J 1991 The application of genetic algorithms to resource scheduling *Proc. 4th Int. Conf. on Genetic Algorithms* ed R K Belew and L B Booker (San Mateo, CA: Morgan Kaufmann) pp 502–8

G1.8 A genetic algorithm for improved computer modeling of mineral separation equipment

C L Karr

Abstract

Recent increases in the computational capabilities available to the mineral industry have led to numerous improvements in the areas of equipment and circuit design, and process control. Many of the novel techniques that can be implemented on today's faster computers rely on models of the mineral processing systems being considered. However, mineral processing systems are often quite complex and are not easily modeled using first principles. As a consequence, empirical computer models are developed from system data. Unfortunately, data collected from industrial systems often contain *outliers* (points that are not consistent with the rest of the data set), and thus can lead to computer models that do not accurately reflect the performance characteristics of the system. Least-median-squares (LMS) curve fitting is a method of robust statistics that guards the process of data analysis from perturbations due to the presence of error in the form of outliers. This procedure has several advantages over classic least-squares (LS) curve fitting, especially in the noisy problem environments addressed by today's process control engineers. Although LMS curve fitting is an effective technique, there are some limitations to the approach. However, these limitations can be overcome by combining the search capabilities of a genetic algorithm with the curve fitting capabilities of the LMS method. This case study presents a procedure for utilizing genetic algorithms in an LMS approach to computer modeling using curve fitting techniques.

G1.8.1 Introduction

A major objective of statistical data analysis is to aid in the extraction and explanation of information contained in a particular data set. Although statistical techniques are used for a wide variety of objectives, many of these objectives are often directly related to computer modeling via curve fitting. Curve fitting receives considerable attention because it plays a key role in a number of engineering endeavors. As engineers strive to take full advantage of the advances in computing power for computationally cumbersome, real-time systems such as numerical modeling, equipment design, and quality control, regression techniques need to be faster and more efficient. Recent advances in artificial-intelligence-based process *control strategies* (Karr and Gentry 1993) have been particularly demanding on traditional statistical techniques because these process control strategies are extremely sensitive to the accuracy of the information extracted by the choice of a curve fitting technique. F1.3

Classical statistical procedures such as LS curve fitting have been used for both the extraction and the explanation of data across a broad spectrum of application domains. However, classical statistical techniques can be quite sensitive to outliers, and, as with other industrial systems, data gathered from separation systems often contain a substantial number of outliers. A variety of robust statistical techniques have been developed that attempt to overcome sensitivity to outliers (Huber 1981). The basic virtue of these robust statistical methods is that small deviations from the model assumptions caused by the presence of outliers hinders their performance only slightly whereas larger deviations do not cause the methods to

fall apart altogether. Generally, these robust methods rely on iterative techniques wherein outliers are identified and delegated to a diminished role. Unfortunately, the problem of outlier detection can be computationally demanding.

Recently, a number of researchers have attempted to produce robust regression techniques that overcome the problem of outlier detection. Several effective techniques have been developed by altering the basic LS approach to curve fitting. LS curve fitting consists of minimizing the sum of the squared residuals; a residual is the difference between the actual data value and the value predicted by the model resulting from the curve fitting technique. The LMS approach (Rousseeuw 1984) replaces the sum of the squared residuals with the median of the squared residuals, thereby yielding an approach that is particularly effective at managing outliers. For instance, in LS curve fitting one outlier can dramatically affect the accuracy of the result. However, in LMS curve fitting up to 50% of the data points can be outliers without degrading the accuracy of the resulting model because only the median residual value is considered. Since the LMS method can withstand the presence of up to 50% of outliers, the method is said to have a *breakdown point* of 50%. Despite the effectiveness of the LMS method, it can face three problematic situations: (i) when there are a large number of data points or when there is a large number of coefficients the method tends to be slow, (ii) when the speed is important, the accuracy of the method suffers due to some approximations that must be made, and (iii) nonlinear curve fitting problems are difficult for the LMS method.

The problems associated with large data sets and nonlinear curve fitting are not unique to the LMS method. In fact, these issues are apparent, to a lesser extent, in LS curve fitting. Researchers have acknowledged and addressed these issues in a number of different ways. A promising approach to tackling the problems associated with performing regression on large data sets and with using nonlinear models is to combine the search capabilities of a *genetic algorithm* with the curve fitting capabilities of LS. In fact, this approach has been shown to be effective in the fine-particle separation industry (Karr *et al* 1991). B1.2

Because of their robust search capabilities and their rapid convergence characteristics, genetic algorithms can be used to search for the constants needed in an LMS curve fitting problem. The genetic algorithm exhibits properties that are desirable in the complex search associated with selecting LMS parameters. The LMS curve fit can often be achieved more rapidly using a genetic algorithm, and the solution is more accurate in certain problems. This section presents two examples in which a genetic algorithm LMS approach is used to develop computer models of separation equipment. Further details of the LMS approach can be found in the article by Karr and Weck (1996).

G1.8.2 Application of genetic algorithm: least median squares to hydrocyclone modeling

This section provides an example of genetic algorithm-LMS curve fitting for the modeling of a separation system (hydrocyclone). This example demonstrates some of the problems associated with conventional LS and LMS curve fitting algorithms, and brings to light some of the strengths of genetic algorithm curve fitting. Hydrocyclones are commonly used for classifying slurries in the mineral processing industry and are used extensively to perform other separations. They are continuously operating separating devices that utilize centrifugal forces to accelerate the settling rate of particles. They are popular because they are simple, durable, and relatively inexpensive. Hydrocyclones are now used increasingly in closed-circuit grinding, desliming circuits, degritting procedures, and thickening operations (Willis 1979). A schematic of a typical hydrocyclone appears in figure G1.8.1.

Hydrocyclones have traditionally been modeled using empirical relationships. Plitt (1976) identified a model to predict the d_{50} or *split size* that is still used extensively today. The split size is that size particle (given by diameter of the particle) that has an equal chance of exiting the hydrocyclone either through the underflow or the overflow, and is often used to quantify a separation process. The model has the form:

$$d_{50} = \frac{C_1 D_c^{C_2} D_i^{C_3} D_o^{C_4} \exp[C_5 \phi]}{D_u^{C_6} h^{C_7} Q^{C_8} \rho^{C_9}} \tag{G1.8.1}$$

where D_c is the diameter of the hydrocyclone, D_i is the diameter of the slurry input, D_o is the diameter of the overflow, D_u is the diameter of the underflow, h is the height of the hydrocyclone, Q is the volumetric flow rate into the hydrocyclone, ϕ is the percent solids in the slurry input, and ρ is the density of the solid. The empirical constants, C_1 through C_9, are selected so that the model accurately matches data that have been collected from the actual hydrocyclone separator being modeled.

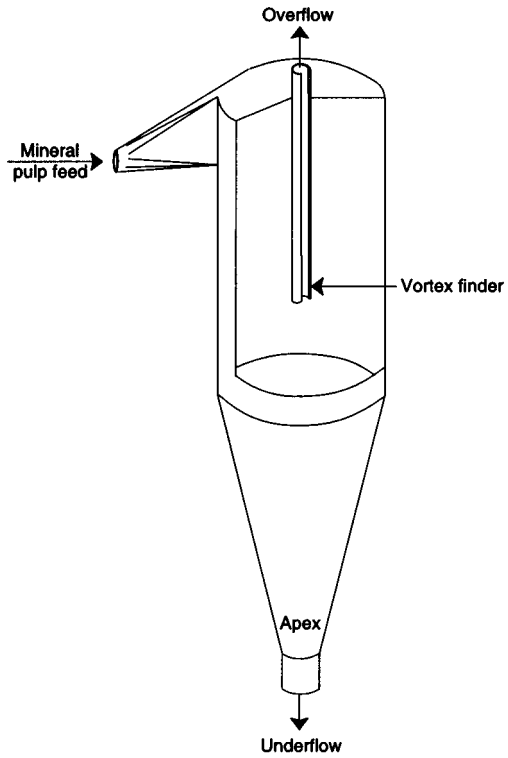


Figure G1.8.1. A typical hydrocyclone receives a mineral slurry and performs a separation: one mineral species exits with the overflow and other mineral species exit with the underflow.

Genetic algorithms often use codings to represent the natural parameter set of the problem; often the parameters are coded as a finite string of characters. Although many character sets have been used in real-world examples (Davis 1991, Goldberg 1989), the parameter sets in this study are coded as strings of zeros and ones. For example, there are nine constants needed to define the hydrocyclone model, and each constant is easily represented as a *binary string*. Eleven bits are allotted for defining each constant (although C1.2 fewer or more bits can be used). These eleven bits are interpreted as a binary number (01001010111 is the binary number 599). This value is mapped linearly between some user-determined minimum ($C_{\min}$) and maximum ($C_{\max}$) values according to the following:

$$C_1 = C_{\min} + \frac{b}{2^M - 1}(C_{\max} - C_{\min}) \tag{G1.8.2}$$

where b is the integer value represented by an M-bit string. The values of $C_{\min}$ and $C_{\max}$ in a given problem are selected by the user based on personal knowledge of the problem. In the problem of developing an LMS computer model of a hydrocyclone separator, a conventional LS curve fit can be performed to approximate the values of the constants. This same form is used to represent the remaining eight constants, and the nine 11-bit strings are concatenated to form a single 99-bit string representing the entire parameter set (C_1 and C_2).

The question of how to evaluate each string must now be addressed. In LS curve fitting problems, the objective is to minimize the sum of the squares of the distances between a curve of a given form and the data points. Thus, if y is the ordinate of one of the data points, and y' is the ordinate of the corresponding point on the theoretical curve, the objective of LS curve fitting is to minimize the sum of the quantities $(y - y')^2$. This square of the error which is to be minimized affords a good measure of the quality of the solution. However, the genetic algorithm seeks to maximize the fitness when expected-number-control reproduction is used (Goldberg 1989). Thus, the minimization problem must be transformed into a maximization problem. To accomplish this transformation, the error is simply subtracted from a large positive constant so that

$$f = C - \sum_{i=1}^{N} (y_i - y_i')^2 \tag{G1.8.3}$$

yields a maximization problem.

A simple genetic algorithm was used in this application: one that implemented reproduction, crossover, and mutation. *Tournament selection* was used for the reproduction operator. Tournaments were held on 10% of the strings (since the population size was 100, the tournaments were held with 10 strings). Eighty percent of the population was replaced via tournament selection per generation. Standard multipoint crossover was implemented with a crossover probability of 0.85; in this application, each string was cut in three places. Standard mutation was implemented with a probability of 0.01. The details of the genetic algorithm are as follows:

population size	$\Longrightarrow$	100
reproduction operator	$\Longrightarrow$	tournament selection
percent of population replaced via reproduction	$\Longrightarrow$	10%
crossover operator	$\Longrightarrow$	three-point crossover
crossover probability	$\Longrightarrow$	0.85
mutation probability	$\Longrightarrow$	0.01
number of generations	$\Longrightarrow$	100.

Figure G1.8.2 summarizes the results of the genetic algorithm's search for appropriate constants to be used in the hydrocyclone model. In this plot, the actual d_{50} size is plotted against the model-predicted d_{50} size. Note that the outliers appear in the lower ranges of the hydrocyclone d_{50} values. This is due, naturally, to the fact that the smaller split sizes are more prone to large errors in measuring. However, regardless of the outliers, the model provides accurate d_{50} predictions for most of the hydrocyclone separations.

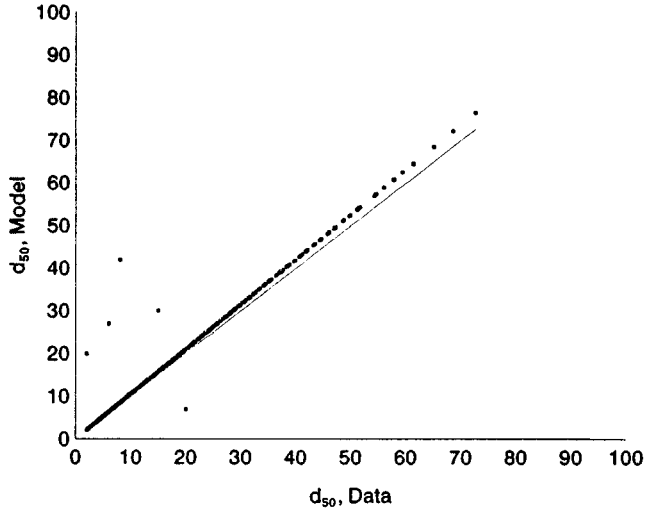


Figure G1.8.2. A genetic algorithm–LMS curve fit to a hydrocyclone separator produces a model that accurately describes the data except for the five outliers existing in the data.

G1.8.3 Application of genetic algorithm: least median squares to grinding circuit

Grinding is a necessary component in the processing of a number of minerals, and improvements in this area could provide substantial cost savings for the minerals industry. Optimizing the process of grinding, however, is a difficult task requiring innovative techniques and accurate computer models. Because of the high energy costs combined with the large-scale use of the process, improvements in the efficiency of the grinding process could have a dramatic economic impact on the minerals industry.

A number of innovative optimization techniques have been developed that are applicable to the improvement of the grinding process (Karr 1993). The success of these optimization techniques requires an effective computer model of the grinding process. Although empirical computer models of the grinding process have been developed (Mehta *et al* 1982), the tuning of these computer models is time consuming and often quite difficult. Thus, using a genetic algorithm for tuning the grinding models is inviting.

The grinding process is characterized by several performance measures, all of which are important in various circumstances. Generally, there are four measures that are especially important indicators of the efficiency of a particular grinding process: (i) *fineness* of the product, (ii) *energy* costs associated with the process, (iii) a *viscosity coefficient*, and (iv) a *viscosity exponent*. Mehta *et al* (1982) have developed an empirical model of grinding that accurately mirrors the response of a coal grinding process. They studied the ability of a simple genetic algorithm to optimize the performance of their computer model of a particular grinding circuit. In the current study, the focus is shifted from using a genetic algorithm to optimize the performance of a tuned model, to employing a genetic algorithm to actually tune the computer model of a grinding process. Only two of the four indicators of grinding efficiency are considered: fineness and energy consumption. However, the steps used to tune the models that predict fineness and energy can be used to tune the models that predict the amount of dispersant addition and the viscosity characteristics. The pertinent modeling equations are

$$\begin{aligned} F = {} & C_1 + C_2 x_{\mathrm{s}} - C_3 x_{\mathrm{b}} + C_4 x_{\mathrm{d}} + C_5 x_{\mathrm{m}} \\ & - C_6 x_{\mathrm{s}}^2 - C_7 x_{\mathrm{b}}^2 + C_8 x_{\mathrm{d}}^2 - C_9 x_{\mathrm{m}}^2 \\ & + C_{10} x_{\mathrm{sb}} + C_{11} x_{\mathrm{sd}} - C_{12} x_{\mathrm{sm}} \\ & + C_{13} x_{\mathrm{bd}} - C_{14} x_{\mathrm{bm}} - C_{15} x_{\mathrm{dm}} \end{aligned} \tag{G1.8.4}$$

and

$$\begin{aligned} E = {} & C_{16} + C_{17} x_{\mathrm{s}} - C_{18} x_{\mathrm{b}} + C_{19} x_{\mathrm{d}} + C_{20} x_{\mathrm{m}} \\ & - C_{21} x_{\mathrm{s}}^2 - C_{22} x_{\mathrm{b}}^2 + C_{23} x_{\mathrm{d}}^2 - C_{24} x_{\mathrm{m}}^2 \\ & + C_{25} x_{\mathrm{sb}} + C_{26} x_{\mathrm{sd}} - C_{27} x_{\mathrm{sm}} \\ & + C_{28} x_{\mathrm{bd}} - C_{29} x_{\mathrm{bm}} - C_{30} x_{\mathrm{dm}} \end{aligned} \tag{G1.8.5}$$

where F is the fineness (or the weight percent less than 38 millimeters), E is the energy in kilowatt hours per ton, x_{s} is the percent solids by weight, $\mathrm{x_b}$ is the maximum ball size, x_{m} is mill speed, x_{d} is dispersant addition, x_{ij} represents the product of x_{i} and x_j, and C_1 through C_{30} are the empirical constants selected so that the model equations accurately reproduce data obtained from a physical system. It is important to note that C_1 through C_{15} can be determined entirely independent of C_{16} through C_{30}. Thus, there are two separate 15-parameter search problems to be solved.

The development of an efficient empirical computer model of a grinding circuit actually depends on tuning 15 parameters that appear in both of the model equations. The very same approach as used in the previous section of employing a genetic algorithm for tuning computer models can be used here. The coding scheme outlined earlier can easily be used, as can the fitness function definition. The details of the genetic algorithm are as follows:

population size	$\Longrightarrow$	100
string length	$\Longrightarrow$	165
reproduction operator	$\Longrightarrow$	tournament selection
percent of population replaced via reproduction	$\Longrightarrow$	10%
crossover operator	$\Longrightarrow$	five-point crossover
crossover probability	$\Longrightarrow$	0.85
mutation probability	$\Longrightarrow$	0.01
number of generations	$\Longrightarrow$	200.

Figures G1.8.3 and G1.8.4 demonstrate the effectiveness of using a genetic algorithm for tuning the empirical constants associated with the grinding models. In these plots, the actual fineness and energy consumption as measured in the physical system are plotted against the computer-predicted values. These figures demonstrate the ability of a genetic algorithm to determine empirical constants for the grinding models.

G1.8.4 Summary

Curve fitting is used in the development of computer models in a wide variety of separation systems. Two approaches to curve fitting are the classical LS and LMS algorithms. However, both of these approaches

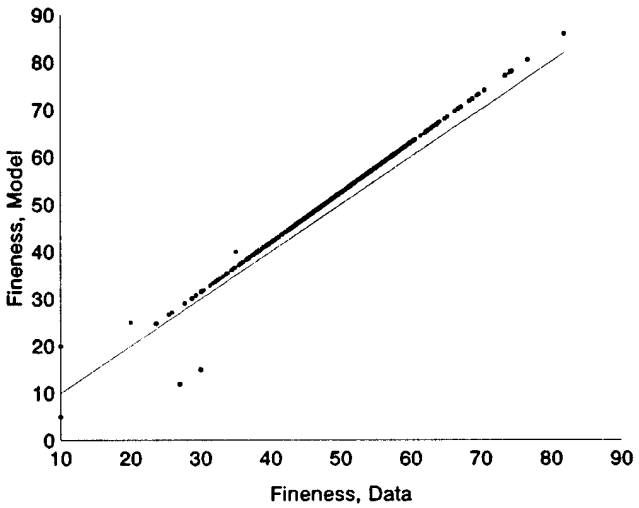


Figure G1.8.3. A genetic algorithm was used to tune the fineness model. Note that a 'perfect' model and 'perfect' data would result in all of the points falling on the 45° line.

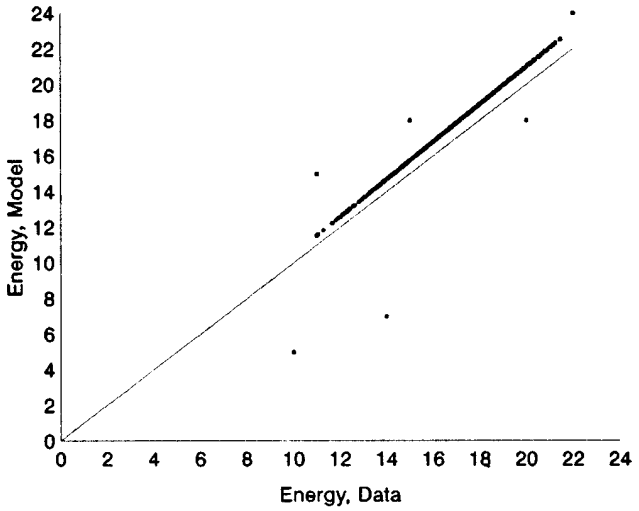


Figure G1.8.4. A genetic algorithm was used to tune the energy consumption model. Note that a 'perfect' model and 'perfect' data would result in all of the points falling on the 45° line.

have their shortcomings. LS methods can be faced with situations for which it is extremely difficult, if not impossible, to solve for the model constants. LMS methods are difficult to apply in nonlinear environments. This chapter has demonstrated the effectiveness of using a genetic algorithm and LMS to model a hydrocyclone separator and grinding.

References

Davis L D 1991 *Handbook of Genetic Algorithms* (New York: Van Nostrand Reinhold)

Goldberg D E 1989 *Genetic Algorithms in Search, Optimization, and Machine Learning* (Reading, MA: Addison-Wesley)

Huber P J 1981 *Robust Statistics* (New York: Wiley)

Karr C L 1993 Optimization of a computer model of a grinding process using genetic algorithms *Beneficiation of Phosphate: Theory and Practice* ed H El-Shall, B Moudgil and R Weigel (Littleton, CO: Society for Mining, Metallurgy, and Exploration) pp 339–45

Karr C L and Gentry E J 1993 Control of a chaotic system using fuzzy logic *Fuzzy control systems* ed A Kandel and G Langholz (West Palm Beach, FL: Chemical Rubber Company) pp 475–97

Karr C L, Stanley D A and Scheiner B J 1991 *A Genetic Algorithm Applied to Least Squares Curve Fitting* Report of Investigations 9339 (Washington DC: US Department of the Interior Bureau of Mines)

Karr C L and Weck B 1996 Improved computer modelling using least median squares curve fitting and genetic algorithms *Fine Particle Separation J.* in press

Mehta R K, Kumar K K and Schultz C W 1982 Multiple objective optimization of a coal grinding process via simple genetic algorithm *Preprint 92–108* Society for Mining, Metallurgy, and Exploration

Plitt L R 1976 A mathematical model of the hydrocyclone classifier *CIM Bull.* **69** 114–23

Rousseeuw P J 1984 Least median of squares regression *J. Am. Stat. Assoc.* **79** 871–80

Willis B A 1979 *Mineral Processing Technology* (Oxford: Pergamon)

G2.1 The use of genetic programming to build queries for information retrieval

Frederick E Petry, Bill P Buckles, Donald H Kraft, Devaraya Prabhu and Thyagarajan Sadasivan

Abstract

Genetic programming is applied to an information retrieval system to improve Boolean query formulation via relevance feedback. Documents are viewed as vectors in term space. A Boolean query is a chromosome in the genetic programming sense. Through the mechanisms of genetic programming, the query is modified to improve precision and recall. Relevance feedback is incorporated via user-defined measures over a trial set of documents. The fitness of a candidate query can be expressed as a function of the relevance of the retrieved set. Preliminary results based on a testbed are given. The form of the fitness function has significant effect and the proper fitness functions take into account relevance based on topicality (and perhaps other factors).

G2.1.1 Introduction

With the advent of the 'infobahn', information services will be available to a larger range of clients than heretofore. Some of these clients, for example, stockbrokers, employers, and purchasing agents, will need to access various information sources repeatedly, using essentially the same query for weeks or months. These clients will be willing to spend part of a day developing a good query.

Here, an application of a variation of *genetic programming* is described for which the objective is to develop good queries. As indicated above, it is not a method likely to prove useful to a casual user of an information system, say, at a library. However, a user who daily or weekly seeks similar information from a nationwide or worldwide network will need to limit the volume of information retrieved without losing access to that which is the most useful. Thus, the foremost requirement is determining what is relevant. This is a dominant concern of information system users now and we expect that it will become more so as networks and information sources expand. To the extent that it becomes more of an issue, query refinement systems such as proposed here will become more important.

B1.5.1

G2.1.2 Information retrieval systems

One way to consider an information retrieval (IR) system is as (i) a set of records which are identified, acquired, indexed, and stored and (ii) a set of user queries for information that are matched to the index to determine which subset of stored records should be retrieved and presented to the user. We can begin to model the retrieval system by presenting a method to identify and store records (Kraft 1985, Kraft and Buell 1983):

D = a set of records
T = a set of index terms (single words or phrases)
F = the indexing function where $\mathbf{F} : \mathbf{D} \times \mathbf{T} \rightarrow \{0, 1\}$.

An example of the matrix **F** is shown in figure G2.1.1.

F

	t_1	t_2	t_3	t_4
d_1	1	0	1	0
d_2	0	0	1	0
d_3	1	0	0	1
d_4	0	1	1	0
d_5	0	1	1	0
d_6	0	0	0	1

Figure G2.1.1. An illustrative retrieval database with six documents and four terms.

One way to compute **F** is by means of the inverted index (Salton 1989) as follows:

$$I(d_i, t_j) = h(d_i, t_j) \log[N/N(t_j)] \tag{G2.1.1}$$

where $h(d_i, t_j)$ is the number of times term t_j occurs in document d_i, N is the number of documents in the database, and $N(t_j)$ is the number of documents in which term t_j occurs at least once. The values of $I(d_i, t_j)$ can then be normalized to the $[0, 1]$ interval by

$$F(d_i, t_j) = I(d_i, t_j) / \max_{x \in \mathbf{D}}[I(x, t_j)] \tag{G2.1.2}$$

then converted to $\{0, 1\}$ by thresholding. It should be noted that some information is lost in the normalization process. Specifically, the Salton inverted frequency index is open ended and indicates how well a term distinguishes a document compared to other terms that also describe it.

The second component of an IR system is the queries. Let us define

$Q =$ a set of user queries for documents

$a : Q \times T \rightarrow \{0, 1\} = a(q, t) =$ the importance of term t in describing the query q.

It is here that one begins to introduce problems in terms of maintaining the Boolean lattice (Kraft and Buell 1983). Because of this, certain mathematical properties can be imposed on **F**, but more directly on a and on the matching procedures described below. Moreover, there is a problem in developing a mathematical model that will preserve the semantics, that is, the meaning, of the user query. The weight a, called a query weight, can be interpreted as an importance weight, as a threshold, or as a description of the 'perfect' document. Let

$$g : \{0, 1\} \times \{0, 1\} \rightarrow \{0, 1\}. \tag{G2.1.3}$$

Thus, $g(\mathbf{F}, a)$ is the retrieval status value (RSV) for a query q of one term (term t) with query weight a in terms of document d, which has index term weight **F** for the same term t.

The function g can be interpreted as the evaluation of the document in question along the dimension of the term t if the actual query has more than one term. In this case, let

$$e : \{0, 1\}^* \rightarrow \{0, 1\} \tag{G2.1.4}$$

be the RSV for a query of many terms, where each term in the query is evaluated as a single-term query against the document and then the total is evaluated using Boolean logic. This notion of allowing e to be a function of the various g values is based on the criterion of separability (Cater and Kraft 1989).

The Boolean query model is used by virtually all commercial retrieval systems. Use of this model is what distinguishes this work from that of Yang and Korfhage (1993).

G2.1.3 The genetic algorithm paradigm employed

The genetic algorithm (GA) architecture used here is that of the simple GA inspired by Holland (1975) which is described in Section B1.2. One should note that there are other paradigms of evolutionary computation (Fogel 1995, Bäck and Schwefel 1993, Mühlenbein 1991), each of which can be found elsewhere in this handbook. B1.2

The method used for selection is known as *linear ranking*. Note that, in the absence of any selection C2.4.2

pressure, each chromosome would, on the average, participate once as the first parent (and once as the second). The objective of selection is to give fitter chromosomes a better than average chance to serve as parents. In ranking this is done by first ordering the chromosomes by decreasing value of fitness. Each is initially given a target selection rate (TSR) of 1.0. A value, δ, between 0.0 and 1.0 is subtracted from the initial TSR of the least fit chromosome and added to the initial TSR of the most fit. In the experiments to be described, $\delta = 1.0$. The TSR of the chromosomes between the best and worst are then set by linearly scaling the difference between that of the best and worst chromosomes. That is, if the chromosomes are ranked from best to worst according to the index $i \in \{0, 1, \ldots, n-1\}$, then the TSR of the ith chromosome in the ranking is $\mathrm{TSR}_i = (1 + \delta) - [2 \times \delta \times i/(n-1)]$.

The TSR is the relative magnitude of probabilistic selection in a random drawing with replacement. In general, it is a real number so the TSR for each chromosome must be converted to an integer representing the actual number of times the chromosome participates as a parent. The method used (developed by Baker (1987)) is known as statistical universal sampling (SUS). While there are many other selection mechanisms (Bäck and Schwefel 1993, Mühlenbein and Schlierkamp-Voosen 1993), many consider that rank-based selection with SUS is best for simple GAs (Goldberg and Deb 1991, Whitley 1989), although there has been recent evidence (Blickle and Thiele 1995) that tournament selection is better.

G2.1.4 Problem representation and solution

In the majority of the applications of GAs, chromosomes are represented as fixed-length *strings* over the alphabet {0,1} or another low-cardinality set. In a variant known as *genetic programming* (GP) (Cramer 1985, Koza 1991), however, chromosomes are represented as variable-length Lisp s-expressions. In the case considered here, the atoms are terms from the set **T**. Crossover consists of replacing a subexpression of one parent with a subexpression from the other. Consider the chromosomes shown below and assume they have been selected as parents. The subexpressions are numbered from left to right as shown.

C1.2 B1.5.1

(and	**(and**	**t1**	**(or**	**t3**	**t5))**	(or	t4	t2))
1	2	3	4	5	6	7	8	9

(or	(or	t1	t2)	(and	**(or**	**t4**	**t5)**	(and	t3	t6)))
1	2	3	4	5	6	7	8	9	10	11

Now, randomly choose two integers. If 2 and 6 were picked, the subexpressions shown enclosed would be identified. Substitute the designated subexpression in the second parent for the designated subexpression in the first parent and one obtains

(and **(or t4 t5)** (or t4 t2))

This, however, is crossover in theory. How it was accomplished precisely in the experiments will be described after additional background is given.

Mutation is necessary to assure that each generation is connected to the entire search space. Practically speaking, it is necessary to introduce new allelic material into a population to reduce the likelihood that it will stabilize at a suboptimum. To mutate, one can change an operator or a term. Both forms were used in the experiments to be described.

Because the work was exploratory in nature, that is, a proof of principle experiment, two fitness functions were employed, a simpler then a more complex one. The first measured only *precision*. Precision is the percentage of documents, among those retrieved, that are relevant,

$$E_1 = \frac{\sum_{i=1}^{n} r_i g_i}{\sum_{i=1}^{n} g_i} \tag{G2.1.5}$$

where

r_i = one (1) if ith document is relevant, otherwise zero (0)

g_i = one (1) if ith document is retrieved, otherwise zero (0).

Relevancy, r_i, is an independent variable for which the value, we assume, is provided by a user. That is, there exists a function $r : \mathbf{D} \rightarrow \{0, 1\}$. The fitness function for the second series of experiments included a term to measure *recall*. Recall is the percentage of relevant documents actually retrieved. In practice,

recall over a document base consisting of millions of entries is difficult to measure. It can be estimated by using a predetermined truth set, as done here, or by statistical sampling

$$E_2 = \alpha \frac{\sum_{i=1}^{n} r_i g_i}{\sum_{i=1}^{n} g_i} + \beta \frac{\sum_{i=1}^{n} r_i g_i}{\sum_{i=1}^{n} r_i} \tag{G2.1.6}$$

where α and β are arbitrary weights.

At this point, we can return to the crossover operator. The experiments were performed using a GA package, not a GP system. The intention is to port the techniques to a GP system at a later date. This entailed some compromises in the representation of chromosomes and the manner in which crossover was performed.

First, queries were represented in disjunctive normal form—a disjunction of conjunctive clauses. The query operators were *and* and *or*. Terms preceded operators and each operator was represented as a bit. Thus

⟨ t3 t2 t6 t8 t1 ⟩ 1101

is equivalent to the infix form

t3 and t2 and t6 or t8 and t1

which, assuming the ordinary operator precedence, has the prefix disjunctive normal form

(or (and t3 t2 t6) (and t8 t1)).

A limit on the number of terms was necessary (11 and 15 in the experiments to be described), but null terms are allowed. Crossover is performed by replacing a random number of sequential terms from the left (or right) of the first parent with terms in the same loci of the second parent. For example, a random number four and the two parents (with the tic, ˆ , added to show the crosspoint)

⟨ t4 t8 t2 t1 ˆ t9 t6 ⟩ 01110

⟨ t7 t3 t4 t2 ˆ t8 t5 ⟩ 11100

produces the child

⟨ t7 t3 t4 t2 t9 t6 ⟩ 01110.

This description of crossover is not entirely precise. The representation of the expression was encoded in binary strings. The crosspoint could be chosen at any locus in the length. Thus, the crosspoint could be between terms, as illustrated, or within the operator sequence (or even within a term). Mutation was performed by 'bit flipping' and the likelihood of mutation for an individual was the length in bits multiplied by the mutation rate.

In summary, one might suspect the results reported below would be similar to those in an unconstrained GP environment; one should not expect them to be identical.

G2.1.5 Experimental results

The experimental document base consisted of 483 abstracts taken from consecutive issues of the *Communications of the ACM*. A set of 4923 terms were taken from the abstracts by analyzing them using a computer program that employed standard heuristics. From this document base, two test sets were constructed. The first test case consisted of 19 relevant documents that were determined manually. A candidate user was asked to collect from the set those articles that pertained to a particular topic for which the user had no formal query. The second test case consisted of 30 relevant documents that were determined by a separate genetic algorithm and the **F** function using a method suggested by Gordon (1991). One document was designated as an anchor then the cluster of most closely matched documents was found using a GA search.

One additional aspect of the experiments needs to be mentioned. Preliminary testing indicated that randomly selecting terms from the set of 4923 terms to populate queries did not work well. Two new strategies were devised. In the first strategy, 80 percent of the terms chosen to seed the initial population came only from the predetermined relevant documents in the test case. The remaining 20 percent were chosen randomly with a uniform distribution. The second strategy was to seed the initial population exclusively with terms from the relevant documents.

Because we are assuming the existence of a test set (truth data), the information needed to seed the initial population using a similar strategy will always be available. Experience with previous but dissimilar applications (Ankenbrandt *et al* 1990) led us to this approach. Only if the user cannot provide a set of sample documents would this not be practical.

There were other issues to be addressed such as population size, the maximum number of terms to permit per query, rates at which crossover and mutation are applied, and, where applicable, α and β weights for fitness functions. These experiments are not explicitly described but the values determined accompany the results reported below.

The performance of the four sets of experiments using fitness function E_1 is shown in table G2.1.2. A similar set of experiments for fitness E_2 is shown in table G2.1.3. All experiments were stopped after a fixed number of queries had been evaluated (64 000 except the last three rows of table G2.1.3 for which 78 000 was the stopping condition). The numerical values in each row in the tables represents an average over three trials. The tables give the number of nonrelevant and relevant documents retrieved by the best query among all those evaluated. *Elitist* strategy was employed. This means that the best query from each generation was carried over to the next generation without modification. The description of the legend for the results is shown in table G2.1.1.

Table G2.1.1. Legend for results tables.

Popsize	Population sizes were greater than 1500. This was for the normal reasons of genetic diversity. Early trials with small populations tended to stagnate with respect to off-line measures.
Test case	Test case 1 refers to the set of 19 relevant documents that was constructed manually. Test case 2 refers to the set of 30 relevant documents that was constructed using automated methods.
Augmented	If the indicator is 'yes' then the results of the experiment were obtained by seeding the initial population using terms from the relevant documents 80% of the time and other terms 20% of the time. If the indicator is 'no' then only terms from the relevant documents were used to build queries for the initial population.
Nonrelevant documents	Contains the number of nonrelevant documents retrieved by the best query averaged over three trials.
Relevant documents	Contains the number of relevant documents retrieved by the best query averaged over three trials.

Table G2.1.2. Fitness function E_1. crossover rate was 0.8, and mutation rate was 0.02. Maximum number of terms allowed was 11 for test case 1 and 15 for test case 2.

Popsize	Test case	Augmented	Nonrelevant docs.	Relevant docs.
1600	1	yes	28.0	18.0 ($E_1 = 0.39$)
1600	1	no	33.3	18.0 ($E_1 = 0.35$)
1600	2	yes	78.7	27.3 ($E_1 = 0.26$)
1600	2	no	82.0	27.7 ($E_1 = 0.25$)

Table G2.1.3. Fitness function E_2. Crossover rate was 0.8 and mutation rate was 0.02. Maximum number of terms allowed was 11 for test case 1 and 15 for test case 2.

Popsize	Test case	Augmented	α/β	Nonrelevant docs.	Relevant docs.
1600	1	yes	1.20/0.8	9.0	16.0 ($E_2 = 1.44$)
1960	1	no	1.05/0.95	3.3	13.7 ($E_2 = 1.53$)
1960	2	yes	1.30/0.7	76.0	27.3 ($E_2 = 0.98$)
1960	2	no	1.30/0.7	71.7	27.0 ($E_2 = 0.94$)

G2.1.6 Discussion of results

With small populations, the average query fitness tended to be higher at the same generation number but the best query tended to be much worse. Possibly smaller populations that maintain the necessary diversity would be feasible if *island model* methods were employed (Mühlenbein 1991). C6.3

In test case 1, there was one document that was never retrieved by an 'optimum' query. There was no such anomalous document in test case 2. Inspection failed to ascertain why the one document was hard to retrieve. For both E_1 and E_2, the initial population contained a query that was about 65 percent as effective as the final query. Thus the search process improved the result by about 50 per cent.

An interesting phenomenon was that fitness E_2 dramatically improved the results for test case 1 but a similar improvement for test case 2 was not experienced. Recall that the method for constructing the two test cases differed considerably. Beyond this, however, no satisfactory explanation was found.

G2.1.7 Conclusions

Two important conclusions can be drawn from the experiments. The first is that GP is a viable method of deriving good queries. Much more experimentation needs to be done to establish the extent of its utility. The second conclusion is that, to achieve good results, the initial population cannot be seeded with terms chosen from the term space according to a uniform distribution. It is necessary to inspect the truth set and seed the initial population predominantly with terms from relevant documents.

We are yet not convinced of which fitness function is most appropriate. In the testing performed E_2 was clearly superior. It has the further advantage of being independent of the structure and form of the queries that the algorithm generates.

References

Ankenbrandt C A, Buckles B P and Petry F E 1990 Scene recognition using genetic algorithms with semantic nets *Pattern Recog. Lett.* **11** 285–91

Bäck T and Schwefel H-P 1993 An overview of evolutionary algorithms for parameter optimization *Evolut. Comput.* **1** 1–24

Baker J E 1987 Adaptive selection methods for genetic algorithms *Proc. 2nd Int. Conf. on Genetic Algorithms (Cambridge, MA, 1987)* ed J J Grefenstette (Hillsdale, NJ: Erlbaum) pp 14–21

Blickle T and Thiele L 1995 A mathematical analysis of tournament selection *Proc. 6th Int. Conf. on Genetic Algorithms (Pittsburgh, July 1995)* ed L J Eshelman (San Mateo, CA: Morgan Kaufmann) pp 9–16

Buckles B P and Petry F E (eds) 1992 *Genetic Algorithms* (Washington, DC: IEEE Computer Society)

Cater S C and Kraft D H 1989 A generalization and clarification of the Waller–Kraft wish-list *Inform. Processing Management* **25** 15–25

Cramer N L 1985 A representation for the adaptive generation of simple sequential programs *Proc. 2nd Int. Conf. on Genetic Algorithms (Cambridge, MA, 1987)* ed J J Grefenstette (Hillsdale, NJ: Erlbaum)

Fogel D B 1995 *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence* (Piscataway, NJ: IEEE)

Goldberg D E and Deb K 1991 A comparative analysis of selection schemes used in genetic algorithms *Foundations of Genetic Algorithms* ed G Rawlins (San Mateo, CA: Morgan Kaufmann) pp 69–93

Gordon M D 1991 User-based document clustering by redescribing subject descriptions with a genetic algorithm *J. Am. Soc. Information Sci.* **42** 311–22

Holland J H 1975 *Adaptation in Natural and Artificial Systems* (Ann Arbor, MI: University of Michigan Press)

Koza J R 1991 A hierarchical approach to learning the Boolean multiplexor function *Foundations of Genetic Algorithms* ed G Rawlins (San Mateo, CA: Morgan Kaufmann) pp 171–92

Kraft D H 1985 Advances in information retrieval: where is that /#*%@^ record? *Advances in Computers* vol 24, ed M Yovits (New York: Academic) pp 277–318

Kraft D H and Buell D A 1983 Fuzzy sets and generalized Boolean retrieval systems *Int. J. Man–Machine Studies* **19** 45–56

Mühlenbein H 1991 Evolution in space and time—the parallel genetic algorithm *Foundations of Genetic Algorithms* ed G Rawlins (San Mateo, CA: Morgan Kaufmann) pp 297–316

Mühlenbein H and Schlierkamp-Voosen D 1993 Analysis of selection, mutation, and recombination in genetic algorithms *Neural Network World* **3** 907–33

Salton G 1989 *Automatic Text Processing: the Transformation, Analysis, and Retrieval of Information by Computer* (Reading, MA: Addison-Wesley)

Whitley D 1989 The GENITOR algorithm and selection pressure: why rank-based allocation of reproductive trials is best *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, June 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 116–21

Yang J-J and Korfhage R R 1993 Query optimization in information retrieval using genetic algorithms *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 603–11

G2.2 A genetic algorithm for the generation of equifrequently occurring groups of attributes

Peter Willett

Abstract

The identification of groups of characteristics with approximately equal frequencies of occurrence is of importance in several areas of information science. This case study describes the use of a genetic algorithm (GA) for the identification of such groups. Experiments with several text dictionaries show that the GA is able to generate groups with a high degree of equifrequency; however, the results are inferior to those produced by an existing, deterministic algorithm if the characteristics are ordered in some way.

G2.2.1 Project overview

Statistical analyses of many types of bibliographic entity show that their frequencies of occurrence all follow a well-marked, near-hyperbolic distribution (Wyllys 1981, Zipf 1949). Examples of this behavior include the numbers of papers published by different authors, the numbers of citations to different papers, the lengths of posting lists in inverted-file retrieval systems, and the occurrences of characters and character substrings in natural language texts. Simple information theoretic considerations suggest that such distributions will limit the efficiency with which information can be stored and retrieved (Lynch 1977, Zunde 1981), and much work has thus been undertaken with the aim of producing sets of characteristics with equal, or at least less disparate, frequencies of occurrence. The resulting sets have been used for the generation of bitstrings for text signature searching, for the compression of natural language texts, for the sorting of dictionaries, and for the generation of monograph identifiers for document delivery systems, *inter alia* (see, e.g. Cooper *et al* 1980, Cooper and Lynch 1984, Goyal 1983, Schuegraf and Heaps 1973, Williams and Khallaghi 1977, Yannakoudakis and Wu 1982). Concepts of equifrequency have also been used in the selection of access paths for numeric database management systems (Motzkin and Williams 1988) and they play a central role in the design of substructural indexing systems for databases of chemical molecules (Ash *et al* 1991).

The work described in this paper was carried out as part of a 2.5 person-year research project, funded by the British Library Research and Development Department, to evaluate the use of *genetic algorithms* B1.2 (GAs) for a range of problems in information retrieval. Three main applications were studied in this project: (i) the creation of nonhierarchic document classifications, (ii) the selection of optimal weights for the indexing of query terms in ranked-output retrieval systems, and (iii) the selection of equifrequent groups as discussed below. Full details of the work are presented by Robertson and Willett (1994, 1995) while other applications of GAs in *information retrieval* are described by Gordon (1988), Petry *et al* G2.1 (1993), and Yang *et al* (1993), *inter alia*.

G2.2.2 Design process

Motivation for an evolutionary solution. There are two obvious ways of dividing up a file of entities with associated frequencies of occurrence to produce sets of equifrequent groupings: in the first method the order of the original file is not preserved (assuming that it was originally ordered in some meaningful way), while the second method partitions a previously sorted input file (such as an alphabetically ordered

dictionary), that is, the file is divided into groups while preserving the original order. These two approaches will be referred to subsequently as *division* and *partition*, respectively, and are illustrated by the following example.

Consider a set of seven objects with frequencies 5, 7, 4, 6, 3, 10, and 5. It is possible to divide this set into four groups with perfect equifrequency, since the groups {5, 5}, {10}, {6, 4}, and {7, 3} all have frequencies summing to ten. But it is not possible to achieve perfect equifrequency in the present case if the frequencies are partitioned: for example, one possible set of groups, given the initial ordering above, is {5, 7}, {4, 6}, {3, 10}, and {5}, with the sums of the groups being 12, 10, 13, and 5, respectively. Thus, a divisive procedure that was able to test all of the possible partitionings for all of the possible orderings of the seven objects in this data set would be able to identify that ordering and that partition that optimized the equifrequency criterion. The partitioning procedure, conversely, can only generate possible partitions derived from the single ordering that is presented to it and is thus far less likely to be able to identify an equifrequent grouping of the frequencies.

The greater simplicity of partitioning means that several partitioning algorithms have been suggested for the identification of equifrequent groupings (see, e.g. Cooper *et al* 1980, Cringean *et al* 1990, Schuegraf and Heaps 1973). Division algorithms are far less common, and appear to have been studied in an information retrieval context only by Yannakoudakis and Wu (1982). Their algorithm involves an initial allocation of frequencies to groups followed by a heuristic procedure that searches through all possible moves of the individual frequencies from each group to all other groups to find those that most increase the equifrequency of the partition. The procedure is extremely time consuming and can be used only when there are limited numbers of frequencies and groups: using frequency data from over 30 000 records in the *British National Bibliography* the experiments of Yannakoudakis and Wu involved dividing the 26 letters of the English alphabet into between 4 and 20 groups and dividing the 244 MARC record subfields into between 5 and 44 groups. The work reported here was carried out to determine whether the novel characteristics of the GA might enable the development of a divisive procedure that was able to process larger data sets than can be encompassed by conventional deterministic algorithms for this purpose.

General description of the type of EA used. The work involved a GA, which was tested with a wide range of parametrizations.

Representation description. The input to the program is a file of N frequencies, each of which denotes the number of times that a specific word in an N-element dictionary occurs in a database.

The frequencies are read into an integer array of length N. An analogous N-element integer array is created to hold the number of the group (in the range $[1, n]$ for a set of n groups) to which each of the frequencies has been assigned. The first n frequencies are assigned, one to each group, and each of the remaining frequencies is then assigned to the group with the smallest current total (thus ensuring that each group is assigned at least one value, for $N \geq n$). The Ith element of these two arrays thus contains the occurrence frequency of the Ith word and the group to which that Ith word has been allocated. Once the initial chromosome has been created in this way, the other chromosomes in the first generation are created by random rearrangement of the second array (i.e. that giving the group membership of each frequency in the input data set). The genetic operators are then applied to the array of group numbers.

Fitness function. Two measures of equifrequency were used as the fitness function. The first was the *relative entropy* (Lynch 1977, Yannakoudakis and Wu 1982). If there are to be n groups such that each group contains *P(I)* occurrences, then the total number of occurrences, *total freq*, is given by

$$total\ freq = \sum_{I=1}^{n} P(I);$$

the relative entropy, H_R, is calculated from

$$H_R = \frac{-1}{\log_2 n} \sum_{I=1}^{n} \frac{P(I)}{total\ freq} \log_2 \frac{P(I)}{total\ freq}.$$

The second fitness function was *Pratt's measure of class concentration* (Carpenter 1979). Let q be defined by

$$q = \sum_{I=1}^{total\ freq} \frac{I \times P(I)}{total\ freq}$$

where the groups are ranked by decreasing frequency and $P(I)$ here is the frequency of the Ith ranked group. Then Pratt's measure, C, is given by

$$C = \frac{(n+1) - 2q}{n-1}.$$

Both the relative entropy and Pratt's measure can have values between zero and unity, with perfect equifrequency being denoted by one and zero, respectively.

These two measures were evaluated using the frequency and group information in the two N-element integer arrays described above. For example, if the relative entropy was being used, $P(I)$, that is, the sum of the frequencies of all of the words allocated to the Ith group, would be calculated for each group I $(1 \leq I \leq n)$, thence giving first *total freq* and then H_R.

Reproductive system. An operator-based GA was used, with equal weights for all of the operators. Generational replacement without duplicates was employed, with 60% of a new generation being created by application of the crossover operators. Parents for crossover were selected either by using *roulette wheel selection* for both parents or by using roulette wheel selection for one and random selection for the other. The remaining members of the new population were selected from the previous population and copied over unchanged into the new generation. The mutation operators were then applied to the resulting sets of chromosomes: the mutation rates varied between 0.8% and mutating all of the remaining 40% that were not created as a result of the application of the crossover operator. C2.2

An elitist strategy was used in some of the experiments to ensure the retention of the fittest chromosome in each generation.

The fitness values were normalized by windowing from the least-fit member of the population or from one standard deviation below the average fitness. Experiments were also carried out in which no normalization was used: there was little difference between the various sets of results.

Operators. In this application, the elements of a chromosome must consist of all and only the members of the discrete set of input frequencies, and it is thus necessary to ensure that the genetic operators yield only valid chromosomes. *One-point*, *two-point*, *order-based*, and *position-based* crossover were used, together with scramble sublist and inversion mutation. The order-based crossover operator was implemented as follows: (i) a binary template was generated randomly with the template values of 0 and 1 resulting in the copying of elements from either the first-parent chromosome or the second-parent chromosome, respectively; (ii) the remaining elements from a child's parent are then copied after reordering to match the order in the other parent. The inversion mutation operator was implemented by selecting two positions in a chromosome at random and then exchanging the values at those positions. C3.3.1

Constraints. The principal constraint was the need to ensure that all, and only, the frequencies in the input file were encoded in each of the chromosomes. This was enforced by the encoding mechanism and by using mutation operators that could not change a chromosome element to a value outside of the range [1, n], which would correspond to a nonexistent group.

Use of domain knowledge and hybrid methods. None.

G2.2.3 Development and implementation

Other methods investigated. The GA approach to the generation of equifrequent sets of characteristics was compared with the performance of the partitioning algorithm of Cringean *et al* (1990). This two-stage algorithm involves an initial, and approximate, partitioning of a dictionary that is then followed by an iterative refinement procedure, in which the initial groupings are merged or split to obtain as high a degree of equifrequency as possible. It was used for comparative purposes since it is known to perform well with a range of types of datum and since it is applicable to data sets of any size.

Practical aspects. The performance measure was the best fitness (as calculated using either the relative entropy or Pratt's measure) amongst all of the chromosomes in the final generation of a GA run.

The termination condition was a fixed number of generations, typically 100 (although some runs were carried out with thresholds of 250 and 500 generations). In fact, the final fittest chromosome was normally identified within 50 iterations, and often very much sooner.

Sources. All of the code was written in C, taking as a basis the routines in standard texts (Davis 1991, Goldberg 1989).

Development platform and tools. The work was carried out on a standard Unix workstation.

G2.2.4 Results

The experiments used four sets of words and their associated frequencies of occurrence in several different types of full text (so as to test the generality of the GA). These were: (i) 3769 words derived from eight English language library and information science papers written by members of the author's department in the University of Sheffield; (ii) 13 033 words derived from three novels in Turkish; (iii) 9781 words from a Turkish language library and information science conference proceedings; (iv) 29 986 words from the *Eighteenth-Century Short-Title Catalogue*. These data sets will be referred to as A, B, C, and D, respectively.

An initial set of experiments was carried out with data set A to ascertain the effects of the many parameters of the algorithm (whether to normalize the fitness values, which crossover operator to use, and what population size to use, *inter alia*) on the fitness values obtained in the final generation. The nondeterministic nature of a GA means that different runs will result in different sets of final chromosomes, and thus in different best values for the relative entropy and for Pratt's measure. Accordingly, each combination of parameter values was run ten times; there was normally very little variation between the ten best values, which implies that the GA is not crucially affected by the essentially random nature of the processing.

As a result of these preliminary experiments, one-point crossover, nonnormalized fitness values, 100-member populations and elitism were used in the main experiments, the results of which are shown in table G2.2.1. Here, parts *(a)* and *(b)* correspond to the use of the relative entropy and Pratt's measure, respectively, as the fitness function, and the bracketed values are those obtained using the deterministic partitioning algorithm of Cringean *et al* (1990). It will be seen that the results in part *(a)* of the table are equal, or nearly so, to the relative entropies obtained when the Cringean algorithm was used to partition the same dictionaries. However, part *(b)* shows that the GA results are noticeably inferior when Pratt's measure is used as the fitness function, except in the case of data set D. We have not been able to identify the reason for this behavior.

The most striking differences between our algorithm and that of Cringean *et al* (1990) are in the memory requirements and running times of the two algorithms. The Cringean algorithm only requires the array of frequencies to be held in memory, while our algorithm requires that the arrays of frequencies and allocated group numbers be held in memory for each of the chromosomes in the population, that is, a population of 50 chromosomes will require 100 times as much memory. An examination of the CPU times for the Unix workstation used in our experiments showed that the CPU was idle for approximately 90% of the time while the data were paged into, and out of, memory. This resulted in very long running times: in the case of the larger data sets, the program had to be run overnight on the Evans and Sutherland ESV 30 workstation that was used in the experiments. By contrast, the Cringean algorithm executed within a very few seconds even for the largest data sets.

The results have focused on the best fitnesses that were obtained. The worst fitnesses in the final populations were also noted, and there was often a large difference between these two values. For example, the best Pratt values for data set D were 0.33, 0.39, and 0.46 for 128, 256, and 512 groups, respectively, while the corresponding worst values were 0.54, 0.59, and 0.65, respectively. The lack of convergence within a population is not too surprising given the great length of the chromosomes used here, when compared with the population sizes that were tested.

Table G2.2.1. Mean best fitness values (to two decimal places) of the relative entropy (*a*) and of Pratt's measure (*b*) for the equifrequency of groups generated using a GA. The bracketed values are those obtained using the partitioning algorithm of Cringean *et al* (1990).

(*a*)	Number of groups		
Data set	128	256	512
A	0.94 (0.94)	0.92 (0.92)	0.88 (0.88)
B	0.99 (0.99)	0.98 (0.99)	0.98 (0.97)
C	0.99 (0.99)	0.98 (0.98)	0.97 (0.96)
D	0.92 (0.92)	0.89 (0.89)	0.86 (0.86)

(*b*)	Number of groups		
Data set	128	256	512
A	0.35 (0.26)	0.42 (0.35)	0.52 (0.46)
B	0.15 (0.06)	0.20 (0.11)	0.25 (0.16)
C	0.18 (0.09)	0.22 (0.14)	0.28 (0.20)
D	0.33 (0.32)	0.39 (0.39)	0.46 (0.46)

G2.2.5 Conclusions

The experimental results demonstrate that the GA described here can divide data sets containing large numbers of frequencies into groups with a high degree of equifrequency, whereas the previous divisive algorithm (Yannakoudakis and Wu 1982) is limited to very small data sets. The algorithm thus provides an effective way of processing large unordered data sets (in fact, it is irrelevant to this algorithm whether the data is ordered or unordered). However, a partitioning algorithm should be used when the data set is ordered; not only are such algorithms at least as effective (as is evidenced by the results in the table G2.2.1) but they are also far more efficient in their use of computing resources (since the Cringean algorithm is very much faster and requires far less memory than does the GA).

However, if a divisive approach is to be taken, then this algorithm would appear to be superior to those that have been reported previously. Apart from the Yannakoudakis–Wu algorithm, reference should also be made to the work of Jones and Beltramo (1991), who describe the application of a GA to what they call the *equal-piles* problem. They tested three encoding methods and a number of crossover and mutation operators, making a total of nine distinct GAs in all. However, their test data set contained just 34 frequencies (33 of which were distinct) that could be divided into ten completely equifrequent groups. Experiments here showed that very large numbers of combinations of parameter values for our GA gave very high relative entropies with this data set, which suggests that it is not very appropriate as a testbed for the evaluation of a GA. The experiments described in this paper therefore provide the best evidence to date for the use of GAs for the identification of equifrequent groupings by divisive means.

References

Ash J E, Warr W A and Willett P (eds) 1991 *Chemical Structure Systems* (Chichester: Ellis Horwood)

Carpenter M P 1979 Similarity of Pratt's measure of class concentration to the Gini index *J. Am. Soc. Information Sci.* **30** 108–10

Cooper D, Dicker M E and Lynch M F 1980 Sorting of textual data bases: a variety generation approach to distribution sorting *Inform. Processing Management* **16** 49–56

Cooper D and Lynch M F 1984 The use of binary trees in external distribution sorting *Inform. Processing Management* **20** 547–57

Cringean J K, Pepperrell C A, Poirrette A R and Willett P 1990 Selection of screens for three-dimensional substructure searching *Tetrahedron Comput. Methodol.* **3** 37–46

Davis L (ed) 1989 *Handbook of Genetic Algorithms* (New York: Van Nostrand Reinhold)

Goldberg D E 1989 *Genetic Algorithms in Search, Optimization and Machine Learning* (Wokingham: Addison-Wesley)

Gordon M 1988 Probabilistic and genetic algorithms for document retrieval *Commun. ACM* **31** 1208

Goyal P 1983 The maximum entropy approach to record abbreviation for optimal record control *Inform. Processing Management* **19** 83–5

Jones D R and Beltramo M A 1991 Solving partitioning problems with genetic algorithms *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, July 1991)* ed R K Belew and L B Booker (San Mateo, CA: Morgan Kaufmann) pp 442–9

Lynch M F 1977 Variety generation—a re-interpretation of Shannon's mathematical theory of communication and its implications for information science *J. Am. Soc. Inform. Sci.* **28** 19–25

Motzkin D and Williams K 1988 A generalized database directory for nondense attributes *Inform. Processing Management* **24** 161–71

Petry F E, Buckles B P, Prabu D and Kraft D H 1993 Fuzzy information retrieval using genetic algorithms and relevance feedback *ASIS '93: Proc. 56th ASIS Ann. Meeting* ed S Bonzi (Melford, NJ: ASIS) pp 122–5

Robertson A M and Willett P 1994 Generation of equifrequent groups of words using a genetic algorithm *J. Documentation* **50** 213–32

——1995 *Use of Genetic Algorithms in Information Retrieval* Report to the British Library Research and Development Department

Schuegraf E J and Heaps H S 1973 Selection of equifrequent word fragments for information retrieval *Inform. Storage Retrieval* **9** 697–711

Williams P W and Khallaghi M T 1977 Document retrieval using a substring index *Comput. J.* **20** 257–62

Wyllys R E 1981 Empirical and theoretical bases of Zipf's law *Library Trends* **30** 53–64

Yang J J, Korfhage R R and Rasmussen E M 1993 Query improvement in information retrieval using genetic algorithms—a report on the experiments of the TREC project *1st Text Retrieval Conf. (TREC-1)* ed D K Harman (Washington, DC: National Institute of Standards and Technology) pp 31–58

Yannakoudakis E J and Wu A K P 1982 Quasi-equifrequent group generation and evaluation *Comput. J.* **25** 183–7

Zipf G K 1949 *Human Behaviour and the Principle of Least Effort* (Reading, MA: Addison-Wesley)

Zunde P 1981 Information theory and information science *Inform. Processing Management* **17** 341–7

G2.3 Genetic information learning

Cezary Z Janikow

Abstract

Supervised learning in attribute-based spaces is one of the most popular machine learning problems studied and, consequently, has attracted considerable attention from the evolutionary computation community. The problem studied here is typical—determining optimal symbolic descriptions for a concept, for which positive and negative examples are provided along with an appropriate language. Key difficulties stem from such concept descriptions being sets of elementary descriptions. The approach presented here uses a variable-length representation—each chromosome represents a complete set of these elementary elements. Another difficulty lies in the gap between the abstract variable-length phenotype and the often used binary genotype. This problem is avoided by defining the evolutionary search at the phenotype level. Finally, most other evolutionary approaches suffer from high time complexity. The approach presented in this case study alleviates this problem by utilizing problem specific search operators and heuristics and by precompiling data to facilitate faster evaluations.

G2.3.1 Project overview

G2.3.1.1 Problem description

Supervised concept learning is a fundamental cognitive process that involves learning descriptions of some categories of objects. Precategorized example objects constitute *a priori* knowledge. Acquired descriptions, often in the form of rules, can subsequently be used to both infer properties of the corresponding concepts (characteristic descriptions) or to decide which category new objects belong to (discriminant descriptions).

Table G2.3.1. Attributes and domains.

Attribute	Domain values
***H**eadShape*	***R**ound, **S**quare, **O**ctagon*
***B**ody*	***R**ound, **S**quare, **O**ctagon*
***S**miling*	***Y**es, **N**o*
***Ho**lding*	***S**word, **B**alloon, **F**lag*
***J**acketColor*	***R**ed, **Y**ellow, **G**reen, **B**lue*
***T**ie*	***Y**es, **N**o*

Consider the problem of learning discriminant robot descriptions in an environment using the attributes of table G2.3.1 (subsequently used abbreviations are boldfaced). The objective is to discover the following (unknown) robot description: *HeadShape* is *Round* and *JacketColor* is *Red* or *HeadShape* is *Square* and *Holding* a *Balloon*. This problem is taken from the article by Wnek *et al* (1990). This robot world is very suitable for this kind of experiment, since it is moderately complex to allow comparative study, yet simple enough to be illustrated by the diagrammatic visualization method.

The task is to learn the description while seeing only random samples of the category (positive examples) and random samples of the countercategory (negative examples). There are $3\times3\times2\times3\times4\times2 = 432$ different robots present in this world—84 belong to the category.

G2.3.1.2 Concept description language

The input language serves as an interface between the environment (teacher) and the system. It should minimize data inconsistency. The output language serves as an interface between the system and the application environment. It should maximize descriptive power. If both languages are the same, it is generally easier to describe, verify, and understand processing mechanisms. One language suitable for both the input and the output is VL_1 (for a brief description and further references see Michalski 1983). Variables (attributes) are the basic units having multivalued domains. According to the relationship among different domain values, such domains may be of different types: nominal—unordered sets; linear—linearly ordered sets; structured—partially ordered sets. Relations associate variables with their values by means of selectors (conditions) having the form [*variable relation value*], with the natural semantics. For the '=' relation, which we use here, *Value* may be a disjunction of domain values (internal disjunction). Conjunctions of selectors form complexes (rules). A full description is given by a disjunction of complexes (set of rules).

Given our description language, the sought concept is: [H=R][J=R]$\vee$[H=S][Ho=B].

G2.3.1.3 System architecture

Because of our approach—which uses existing inductive operators as search means, is guided by Darwinian selective pressure and problem heuristics, and uses operators and evolutionary ideas as means to control the search (section G2.3.2)—the overall system architecture closely resembles that of an AI production system. It is illustrated in figure G2.3.1. Population represents the current state of the search-space exploration. This state is modified by applications of Darwinian selection and the inductive operators, which are biased by problem heuristics (section G2.3.2.6).

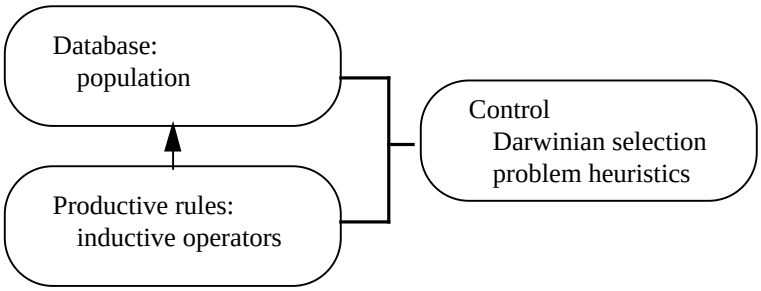


Figure G2.3.1. System architecture.

G2.3.2 Design process

G2.3.2.1 Motivation

Supervised learning is characterized by relatively large search spaces. For our problem, there are $7\times7\times3\times7\times15\times3 = 46\,305$ valid rules (there are $2^3-1=7$ valid selectors for *HeadShape*, etc). This gives about $10^{15\,000}$ different rule sets (i.e., the power set of the number of rules—infinitely many if duplications are not taken care of). This means that no exhaustive method could be used. Whatever the specific objectives, supervised learning also exhibits multimodality. This means that no hill climbing techniques could be used. Two obvious approaches are to use heuristics to guide the search or to use evolutionary techniques to provide more robust search. Known procedural/heuristic methods include rule discovery AQ systems (Michalski *et al* 1986) and decision trees (Quinlan 1986). Evolutionary approaches include *classifier systems* (see e.g. Riolo 1988) and *genetic algorithms* (see e.g. Spears and De Jong 1990). B1.5.2, B1.2 Our objective is to combine the two approaches in an evolutionary method guided by heuristics.

As heuristics, we use inductive operators and refinement bias. Michalski (1983) provides a detailed description of various inductive operators that constitute the process of inductive inference. In the language VL_1, example operators include condition dropping from a rule, adding an alternative rule or dropping a rule, extending a condition, or closing an interval in a linear condition. These operators are either generalizing or specializing existing knowledge. For example, dropping a condition generalizes a rule, while dropping a rule specializes a rule set. Moreover, needed knowledge refinement is said to be heuristically dependent on the current rules. For example, an overgeneralized rule (i.e. covering some negative examples) should be further specialized (e.g. by adding conditions).

Having the operators, evolutionary control (population and Darwinian selective pressure) is used to apply the operators. Application probabilities are further guided by the refinement bias, as described below.

G2.3.2.2 Requirements

There are two requirements guiding our approach: improving speed of other existing genetic algorithms and also producing descriptions of low complexity. Efficiency is addressed by data compilation (section G2.3.3.1) speeding up evaluations, as well as by incorporating heuristics and by using the inductive operators to guide and conduct the search.

Description complexity affects human understanding of the generated knowledge. Following Wnek *et al* (1990), we defined complexity as twice the number of rules plus the total number of conditions. Reducing so-defined complexity also means that no redundant rules should be retained. However, such redundant rules are not removed by explicit mechanisms—Darwinian pressure coupled with proper fitness provides implicit means.

G2.3.2.3 Representation

Each chromosome is capable of representing an unrestricted number of rules. Each rule (complex) is represented by a number of conditions. Because the possible conditions are determined from the problem specific language (such as the attributes of table G2.3.1), each rule can be restricted syntactically (section G2.3.3). For dual categories, a rule does not have to explicitly represent a category designation. Instead, it is assumed that the represented rule set describes the category (a concept), and that anything not covered by this description represents negation of the concept. There are no preset restrictions on the number of rules per chromosome other than physical memory limitations.

G2.3.2.4 Fitness

Fitness must reflect learning criteria. In supervised learning two criteria are typically used: description completeness (positive-example coverage) and consistency (avoidance of negative-example coverage). In an evolutionary algorithm, all objectives are usually combined to form a single fitness value. Accordingly, we define:

$$\text{correctness} = \frac{w_1 \times \text{completeness} + w_2 \times \text{consistency}}{w_1 + w_2}$$

where the two coefficients can be used to bias the search toward more complete or more consistent formulas. Completeness and consistency are defined in table G2.3.2, where e^+ (e^-) is the number of positive (negative) training examples currently covered by a rule, ε^+ (ε^-) is the number of such examples covered by a rule set, and E^+ (E^-) is the total number of such training examples. These two measures are meaningful only to rule sets and individual rules. For conditions, measures of the parent rule are used. These definitions assume a full-memory model (all training examples are retained).

Table G2.3.2. Attributes and domains.

Syntactic structure	Completeness	Consistency
A rule set	ε^+/E^+	$1-\varepsilon^-/E^-$
A rule	e^+/ε^+	$1-e^-/\varepsilon^-$

One of our objectives is to produce descriptions of low complexity. For example, this is necessary to promote dropping redundant rules. Suppose cost denotes normalized measure of complexity. Then, fitness is determined by

$$\text{fitness} = \text{correctness} \times (1 + w_3(1 - \text{cost}))^f \tag{G2.3.1}$$

where w_3 determines the influence of the cost, and f grows very slowly on $[0, 1]$ as the population ages. The effect of the very slowly rising f is that initially the cost influence is very light in order to promote wider space exploration, and it only increases at later stages in order to minimize complexity.

G2.3.2.5 Operators

There are three syntactic levels in the chromosomes: conditions, rules, and rule sets. Different operators apply to different levels. Moreover, some operators specialize, while others generalize the existing knowledge.

Table G2.3.3. Genetic operators.

	Syntactic level		
Knowledge refinement	Rule set	Rule	Condition
Generalization	Rules copy New event Rules generalization	Condition drop Turning conjunction into disjunction	Reference extension
Specialization	Rules drop Rules specialization	Condition introduce Rule-directed split	Reference restriction
Independent	Rules exchange	Rule split	

All the operators are listed in table G2.3.3. For a complete description, see the article by Janikow (1993). Here we illustrate two of them, using our descriptive language and the idea of diagrammatic visualization, which is an extension of the well-known Karnaugh maps (Wnek *et al* 1990). The illustrated operators are

- rules generalization—this operator modifies one chromosome by selecting two random rules and replacing them with their most specific generalization (figure G2.3.2)—and
- reference extension—this operator modifies a single condition of a selected rule. If the condition is a restriction on a linear attribute, closing the reference interval has a higher probability (Michalski 1983). For nominal attributes, selected restrictions on the attribute are dropped. This operator is illustrated in figure G2.3.3.

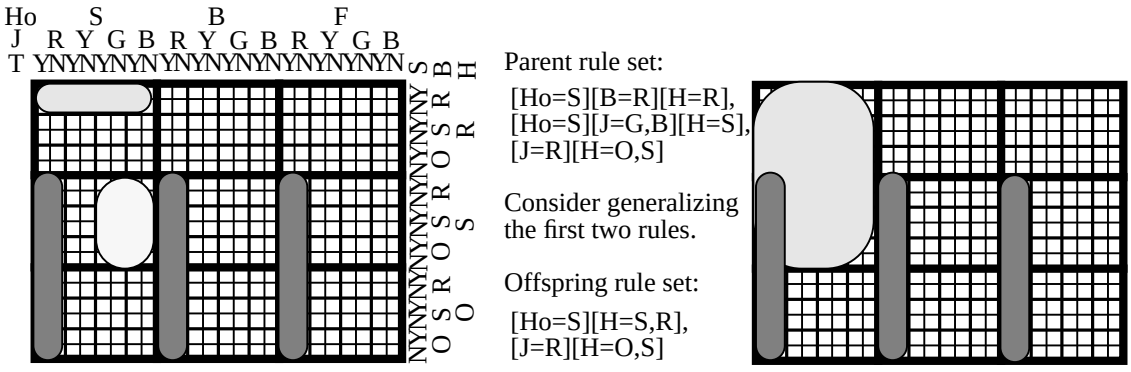


Figure G2.3.2. Illustration of 'rules generalization'. Reprinted from Janikow (1993) by kind permission of Kluwer Academic Publishers.

Actual application probabilities depend on context, such as the current coverage of positive and negative examples. Probabilities of generalizing operators are increased for applications to structures (rule

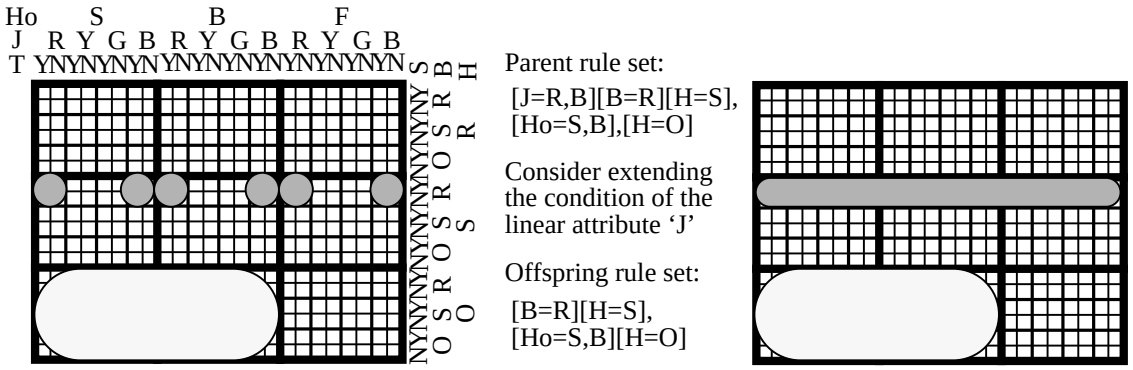


Figure G2.3.3. Illustration of 'Reference extension'.

sets, rules, conditions) that are incomplete, and decreased for those that are inconsistent. On the other hand, probabilities of specializing operators are increased for applications to structures that are inconsistent, and decreased for those that are incomplete. Moreover, the levels of probability increase (decrease) are based on the levels of inconsistency/incompleteness.

The following experimental formulas define the dynamic probabilities for various operators:

- *Generalizing operators.* $p' = p(2/3 - \text{completeness})(1/2 + \text{consistency})$.
- *Specializing operators.* $p' = p(1/2 + \text{completeness})(3/2 - \text{consistency})$.

The new value p' is the adjusted probability, and p is the static probability. It is important to mention that since p' is computed differently for each rule and rule set, it does not replace the *a priori* p. For example, consider a rule set with the following measures: completeness $= 10\%$, consistency $= 90\%$. The generalizing operators have probabilities of application additionally adjusted by $(3/2 - \text{completeness})(1/2 + \text{consistency}) = 1.82$ and the specializing operators have the probabilities adjusted by $(1/2 + \text{completeness})(3/2 - \text{consistency}) = 0.42$. In other words, this particular rule set would face an increasing pressure for generalization. The simplicity of these formulas guarantees a low computational overhead.

The program also uses a mechanism to adjust the probabilities to chromosome sizes. For details, see the article by Janikow (1993).

G2.3.2.6 Reproduction

We use a standard *fitness proportional* generational model: first a new population is selected using the stochastic universal sampling mechanism, and then operators modify chromosomes of the new population. All operators are given static probabilities, which are dynamically adapted as described above. Operators fire in a random order. In this setting, it is possible that a few operators modify the same single chromosome. C2.2

G2.3.2.7 Constraints and domain knowledge

Since our chromosomes represent the phenotype, language specific constraints are explicitly addressed. We did not make any provisions to accommodate additional constraints. However, initial hypotheses can be processed (see the following section).

G2.3.2.8 Initialization

We provide three different initialization routines. For a given run, all three routines can initialize different portions of the population. First, random rules can be generated (given the specific language). Second, if initial hypotheses are present, they may initialize some chromosomes. This way, those hypotheses will be refined as necessary. Finally, sets of a few positive examples could be used as initial chromosomes.

G2.3.2.9 Termination condition

The simulation is set for two stages, each of which is terminated after a preset number of generations.

The first stage is characterized by very low f (G2.3.1), and it is terminated early if a complete and consistent description is found. In the second stage, f grows faster in order to simplify the description. It is never stopped early since it is generally not known what descriptions are sought.

G2.3.3 Development and implementation

G2.3.3.1 Data compilation

A very commonly cited disadvantage of evolutionary algorithms is their time complexity. This problem is aggravated in full-memory systems due to extensive pattern matching. Concerned with such problems, we designed a special method of data compilation, aimed at improving the time complexity of the system.

The idea is as follows: rather than storing training example data in terms of features, store features in terms of data coverage. In other words, for each possible feature, retain information about the examples covered by this feature. This must be done separately for each category, even those not being explicitly learned. This is achieved by enumerating all learning examples, and constructing binary coverage vectors.

The idea behind these vectors is analogous to that of representing conditions. A coverage vector is constructed for both E^+ and E^- separately. In this vector, a binary one at position n indicates that the structure that owns this coverage vector covers the example #n. For example, the vectors in table G2.3.4 indicate that the given feature covers positive events #1, 8, 17, 19 (out of 25), and negative events #12 and 14 (out of 15).

Table G2.3.4. Examples of binary coverage vectors.

Positive coverage vector:	1000000100000000101000000
Negative coverage vector:	000000000001010

During the actual run of the system, similar vectors are constructed and retained for all structures of the population: from the features upwards to rules and rule sets. For example, having the feature coverage vectors we can easily construct both positive and negative coverage of the condition $[B = R, O]$ by means of a simple bitwise OR on coverage vectors of features $(B = R)$ and $(B = O)$. Subsequently, conditions' coverages are propagated to rules by means of bitwise AND. Finally, rules' coverages are propagated to rule sets again by means of bitwise OR.

Perhaps the most important effect of such an approach is that we can incrementally upgrade such coverages using a minimal amount of work after the initial database is fully covered. For example, consider a case of the 'rules copy' operator, which copies selected rules between two chromosomes, applied to the following two rule sets: $R_1 = r_1^1, r_1^2$ and $R_2 = r_2^1, r_2^2, r_2^3$, and suppose the operator copies r_2^2 to chromosome R_1. The coverage of the second rule set does not change. To compute the coverage of the first rule set it is sufficient to perform bitwise OR between the coverage of the rule r_2^2 (which did not change during this operation) with the coverage of the original R_1. In other words, we compute this coverage using two bitwise OR operations (one for the positive and one for the negative coverage).

G2.3.3.2 Internal representation

Each complex is a conjunction of a number of conditions. Each condition is represented with the '=' relation and the internal disjunction. The number of such possible conditions, in a given complex, is bounded by the total number of attributes. We use that bound as a way of simplifying the internal representation: a complex is represented by a vector of conditions. Furthermore, for simplicity and efficiency, we associate a fixed positional correspondence between the attributes of the vector.

The above ideas are illustrated in figure G2.3.4, assuming the attributes of table G2.3.1 on an 8-bit machine and domain enumeration as of table G2.3.1 left to right. The illustration is for a chromosome representing the concept: *HeadShape* is *Round* or *Octagonal* and *Smiling* or *Holding* a *Sword* or a *Balloon* and *JacketColor* is *Red, Yellow* or *Green*.

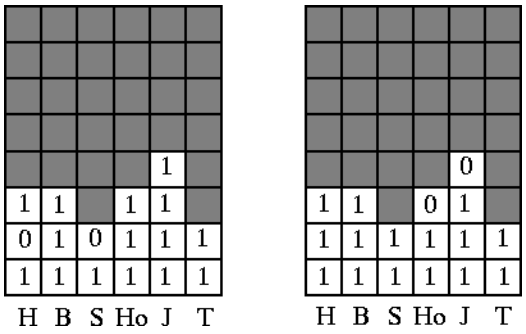


Figure G2.3.4. Internal representation.

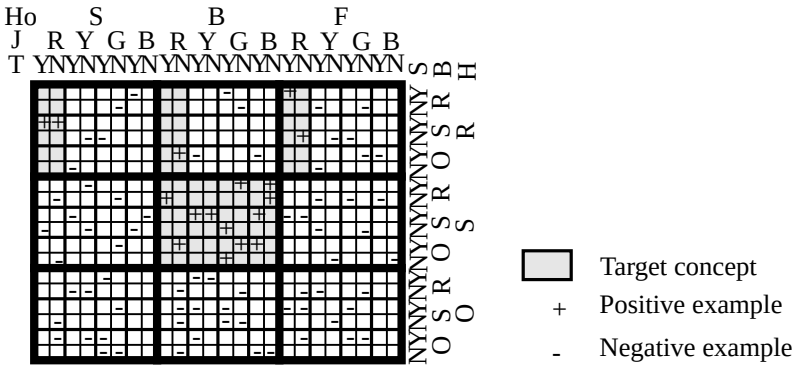


Figure G2.3.5. The goal description and the training examples. Reprinted from Janikow (1993) by kind permission of Kluwer Academic Publishers.

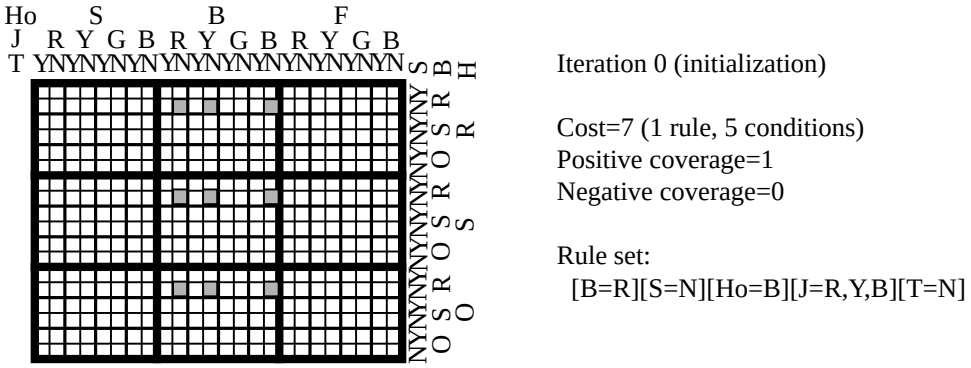


Figure G2.3.6. The best initial chromosome. Reprinted from Janikow (1993) by kind permission of Kluwer Academic Publishers.

G2.3.3.3 Platform

The system is implemented in C, but was only tested on a SUN workstation using 'cc' and SunOS 4.1.1.

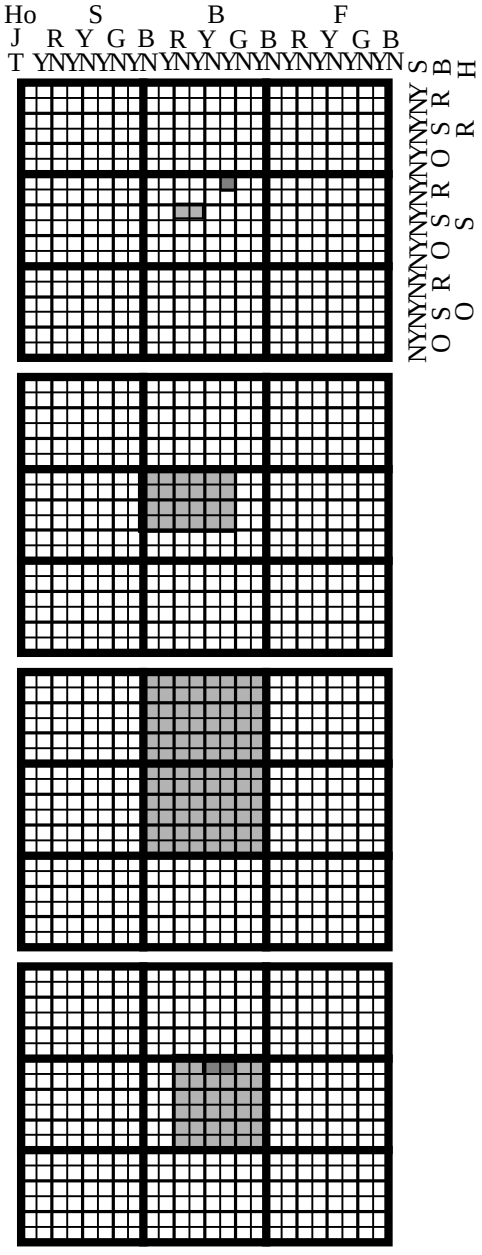
G2.3.4 Results

The population size is set to 40, initialized equally by both random descriptions and positive training events. The system is set to run 100 iterations (equally split into the two stages). Other implementation parameters are set as follows: $w_1 = w_2 = 0.5$, $w_3 = 0.02$, the cost was normalized with respect to the highest cost in the current population.

G2.3.4.1 Illustration

For the illustration, we trace a run using a random 20% of both positive and negative examples: 17 and 70 respectively (see figure G2.3.5 for a visualization of the target concept and the training examples). The best initial chromosome is illustrated in figure G2.3.6, while figure G2.3.7 illustrates the best chromosomes from the first four generations with improvements (there are no improvements in iterations 4 through 6).

This problem proves to be extremely simple for our algorithm. After only 27 iterations, or about 3 CPU seconds on a SPARCstation, a complete and consistent description is found. Moreover, this description matches exactly the desired solution—there are no redundancies, and thus the second iteration stage is externally halted.



Iteration **1**

Cost=15 (2 rules, 11 conditions)
Positive coverage=3
Negative coverage=0

Rule set:
[H=S][B=S][S=Y][Ho=B][J=Y],
[H=S][B=R][S=Y][Ho=B][J=G][T=N]

Iteration 2

Cost=6 (1 rule,4 conditions)
Positive coverage=6
Negative coverage=0

Rule set:
[H=S][B=S,O][S=Y][Ho=B][J=Y,G,B]

Iteration 3 (an overgeneralization)

Cost=4 (1 rule,2 conditions)
Positive coverage=12
Negative coverage=4

Rule set:
[H=R,S][Ho=B]

Iteration 7 (one redundant rule)

Cost=12 (2 rules, 8conditions)
Positive coverage=10
Negative coverage=0

Rule set:
[H=S][Ho=B][J=Y,G,B],
[H=S][B=R][S=Y][Ho=B][J=G]

Figure G2.3.7. First four improvements. Reprinted from Janikow (1993) by kind permission of Kluwer Academic Publishers.

Table G2.3.5. Error rate summary in the robot world.

	Learning scenario (positive%/negative%)				
System	6%/3%	10%/10%	15%/10%	25%/10%	100%/10%
AQ15	22.8%	5.0%	4.8%	1.2%	0.0%
BpNet	9.7%	6.3%	4.7%	7.8%	4.8%
C4.5	9.7%	8.3%	11.3%	2.5%	1.6%
CFS	21.3%	20.3%	21.5 %	19.7%	23.0%
GIL	4.3%	1.1%	0.0%	0.0%	0.0%

Table G2.3.6. Knowledge complexity summary in the robot world.

	Learning scenario (positive%/negative%)				
System	6%/3%	10%/10%	15%/10%	25%/10%	100%/10%
AQ15	2.6/4	1.6/3	1.6/3	1.6/3	1.6/3
BpNet	NR	18/29	NR	NR	32/54
C4.5	6.8/12.2	4.4/9.2	4.8/9.2	4.8/9.2	3.8/7.3
CFS	NR	NR	NR	NR	NR
GIL	1.4/2.6	1.6/3	1.6/3	1.6/3	1.6/3

G2.3.4.2 Comparative results

The other systems used in the experiment (reported by Wnek *et al* 1990) are rule-based AQ15, neural network BpNet, decision tree with rules generator C4.5, and genetic classifier system CFS. Table G2.3.5 reports the average error rate for the five experimental concepts for all five systems while learning one concept at a time (the results of the other four obtained from those published experiments). Surprisingly, our system (GIL) produces the highest recognition rate, especially when seeing only a small percentage of the robots. This result can be attributed to the used simplicity-biased evaluation formula.

Table G2.3.6 reports the average acquired knowledge complexity by listing both the average number of rules and the average number of conditions, as learned by all five systems for different learning scenarios in the same experiment. The NR entry indicates that the complexity was large and not reported in the reference paper. The reason for the higher complexity of the connectionist approach is that this is a nonsymbolic system operating on numerical weights rather than on the problem symbols. On the other hand, the high complexity of the CFS approach can be attributed to the fact that the symbolic processing was being done in the representation rather than the problem space and to the lack of a similar bias for simplicity. This result is rather common for classifier system approaches. Therefore, it is a pleasant surprise to find that GIL's knowledge is at the same complexity level as that of AQ15. This demonstrates one of the system's characteristics: ability to generate easily comprehensible knowledge (measured here by complexity of its VL_1 output). The 1.4/2.6 result in the first column is clearly an oversimplification of the knowledge due to insufficient number of training events (here only about ten negative events were available).

G2.3.5 Summary

We designed the evolutionary algorithm for inductive concept learning with two objectives: accelerated learning and the production of descriptions with low complexity. To accomplish this, we used inductive operators and heuristics, and we used precompiled examples facilitating faster evaluations. To provide grounds for the operators and heuristics, chromosome representation was moved to the phenotype level. In this context, the only evolutionary ideas used were selective pressure and population. Experimental results suggest that we succeeded in accomplishing those objectives.

References

Janikow C Z 1993 A knowledge-intensive genetic algorithm for supervised learning *Machine Learning* **13** 189–228

Michalski R S 1983 Theory and methodology of inductive learning *Machine Learning: an Artificial Intelligence Approach* ed R S Michalski, J Carbonell and T Mitchel (San Mateo, CA: Morgan Kaufmann) pp 83–134

Michalski R S, Mozetic I, Hong J and Lavrac N 1986 *The AQ15 Inductive Learning System: an Overview and Experiments* Technical Report UIUCDCS–R–86–1260, Department of Computer Science, University of Illinois at Urbana-Chapaign

Quinlan J R 1986 Induction on decision trees *Machine Learning* **1** 81–106

Riolo R L 1988 *CFS-C: a Package of Domain Independent Subroutines for Implementing Classifier Systems in Arbitrary, User-defined Environments* Technical Report, Logic of Computers Group, Division of Computer Science and Engineering, University of Michigan

Spears W M and De Jong K A 1990 Using genetic algorithms for supervised concept learning *Proc. 2nd Int. Conf. on Tools for AI (Herdon, VA, November 1990)* ed A Dollas, W T Tsai and N G Bourbakis (Los Alamitos, CA: IEEE Computer Society Press) pp 335–41

Wnek J, Sarma J, Wahab A and Michalski R S 1990 Comparing learning paradigms via diagrammatic visualization *Methodologies for Intelligent Systems 5* ed M Emrich, Z Ras and M Zemankowa (Amsterdam: North-Holland) pp 428–37

G3.1 Genetic programming for stack filters

E Howard N Oakley

Abstract

A range of techniques was used to search for the fittest filter to remove noise from data from a blood flow measurement system. Filter types considered included finite impulse response (FIR), RC (exponential), a generalized FIR form, and stack filters. Techniques used to choose individual filters were heuristic, the genetic algorithm, and genetic programming. The efficacy of filters was assessed by measuring a fitness function, derived from the root mean square error. The fittest filter found was a stack filter, generated by genetic programming. It outperformed heuristically found median filters, and an FIR filter first produced by the genetic algorithm and then improved by genetic programming. Genetic programming proved to be an inexpensive and effective tool for the selection of an optimal filter from a class of filters which is particularly difficult to optimize. Its value in signal processing is confirmed by its ability to further improve filters created by other methods. Its main limitation is that it is, at present, too computationally intensive to be used for on-line adaptive filtering.

G3.1.1 Project overview

Although there are a number of toolkits and techniques for the selection and design of particular classes of filter, there does not appear to be any general method for the development of a filter intended to remove noise from data. This is particularly true when considering a relatively new class of filter, the stack filter, which offers an almost unlimited range of possibilities, and is easily implemented in hardware.

Stack filtering (Wendt *et al* 1986) consists of three stages. Consider a filter which slides a window of width n across input data with integer values in the range 1–m. Data in a given window are first decomposed into a matrix of boolean values, such that an input value of $x = m$ is represented by a column of m cells containing the value 1 below $(n - m)$ cells containing the value 0, within the matrix. Then, the operations that characterize the filter are applied across the rows of the matrix, resulting in a single-column matrix, which is finally recomposed into the single output value by reversing the decomposition stage (summing the 0 and 1 values of the single column).

Any logical and arithmetical operations can be applied in the second stage, including those which result in median filters, a subset of stack filters. For a window of width $n = 2a + 1$ containing data values x_1–x_n, in which the input value x_1 is decomposed into a matrix column x_{11}–x_{1m}, a median filter can be represented as applying the operation

$$\sum_{j=1}^{m} \left(\sum_{i=1}^{n} x_{ij} > a \right)$$

if the result of the inequality is represented as 1 if true, 0 if false. This can in turn can be transformed into a more complicated purely logical form.

This project was undertaken by one person as part of the development of the data processing for a blood flow measurement system which generates noisy estimates of instantaneous blood flow from two sites, integral values from 0 to 255 delivered at a rate of 40 Hz from each site. Downstream of this filtering, the data were to be dissected into packets representing each cardiac cycle for further processing;

this demanded the removal of spikes and troughs of noise, as it required peak detection to perform the dissection. The aim was therefore to select the filter which resulted in the cleanest output signal, so that it could initially be implemented off-line, and later incorporated into a software-based real-time processing system. Although adaptive filtering techniques were considered, it was decided to try a fixed approach in the first instance, employing evolutionary computation for one-time optimization of the filter.

Three other types of filter were deemed worthy of inclusion in this project. The blood flow measurement system provided as standard single-pole RC (exponential) filters with a range of time constants t_c. These make each output value the result of

$$\sum_{n=-\infty}^{t} \frac{e^{-n/t_c}}{\sum e^{-n_t/t_c}} x_n$$

where x_n is the input value at time n and the denominator is the sum of exponential weights. With a single parameter, the time constant, determining the filter, this is particularly easy to optimize by trial and error, and has also historically been easy to implement in hardware (the name referring to the resistance–capacitance analog circuit used in older instruments for this purpose).

Another commonly used type of filter is the finite-impulse-response (FIR) filter, which is essentially a weighted average applied across the window, and can be represented as

$$\sum_{n=t-a}^{t+a} w_i x_n$$

for a window of width $(2a + 1)$ and the same number of weights which sum to 1.0. Such filters often conform to standard patterns described according to the weights used, for example, the 'bell' FIR filter. They are also easy to incorporate into optimization using the genetic algorithm, as this is only required to determine the best performing set of weights (Etter *et al* 1982). Other approaches have also been used to optimize FIR filters, in particular adaline and related forms of neural network (Widrow and Stearns 1985).

The final type of filter examined was a more generalized derivative of the FIR filter, in which the output value is the result of any arbitrary mathematical combination of the n input data values. Although this is not normally used in signal processing because of its complexity, it is easy to incorporate into a scheme of optimization by genetic programming.

G3.1.2 Design process

Three basic techniques were chosen for the development of classes of filter, and to identify the best performing within each class. The first was heuristic, in which the advice of experts within the field of signal processing was canvassed, and coupled with that contained within established texts such as that by DeFatta *et al* (1988). This generated four candidate types: a rectangular and bell FIR filter (weights even and Gaussian respectively), RC filtering as employed by the blood flow measurement system, and median filters with window widths three to nine.

The *genetic algorithm* was used to develop conventional FIR filters with a window width of seven (*7 tap*), by optimizing the *taps* or weights. *Genetic programming* was employed in two different forms. First, it was used to optimize filters of the generic FIR class, by applying the four fundamental arithmetical operators to a terminal set containing the window of seven input data values, including some in which the initial populations were seeded with individual *S-expressions* containing the fittest conventional FIR weights arrived at by the genetic algorithm. Genetic programming was also used to optimize the logical operators for a stack filter across a window of the same width.

B1.2 B1.5.1 C1.6

Although exhaustive search was possible for the special case of median filters, up to a window width of nine, and to a degree for RC filters, it is impractical for any of the other classes of filter employed in this study. For instance, even the simple window width seven FIR filter, using integral weights in the range 0–255, can generate approximately 7.2×10^{16} different filters. If assessed at a rate of 1000 per second, as might be possible on a high-performance computer system, it would take some 2.3 million years to consider every filter. Even when constrained to a particular window width, the generic FIR and stack classes of filter have still less finite numbers of different filters which would require assessment. Previously, Chu (1989) employed the genetic algorithm to optimize stack filters, and enjoyed considerable success although operating in a search field which was more constrained than that possible with genetic

programming. Others, such as Ansari *et al* (1992), have found that neural networks are also effective for this purpose.

Inevitably, the goal was to remove as much noise as possible without distorting the output data. However, as the measurement system has neither a 'gold standard' technique against which its output can be compared, nor any method of injecting known data, this was recognized as being a problem. The only way in which idealized noise free data could be produced was by hand cleaning a noisy example data set, to generate what was considered to be the best estimate of the underlying data. Although this was a subjective step, there is good knowledge of the physiology of blood flow and thus the expected waveform which was to be recovered.

By convention, the performance of filters was assessed in terms of the root mean square (rms) error, where the error is the difference between the predicted and actual values for each data point. This translates conveniently into a raw fitness function of

$$\frac{1}{1+\left[\sum(x_{\mathrm{f}}-x_{\mathrm{c}})^2/n\right]^{1/2}}$$

where x_{f} is the filtered and x_{c} the clean value for a given data point in the test set of size n. This is already standardized with the values 0.0 representing completely unfit and 1.0 perfectly fit filter results. Whilst there are cogent arguments in favour of a fitness function based on the mean absolute error, it was felt that the additional weighting accorded by squaring would help reduce outliers, which could adversely affect the final system.

G3.1.3 Development and implementation

A single input data set consisting of 600 values was used in every run, together with its corresponding 600 hand-cleaned perfect output values. In order to ensure that the noise within the data was representative of that in a large number of data sets, synthetic noise was generated using a mixture of uniform (rectangular) and Gaussian-distributed pseudorandom values, which resembled statistically the noise originally found in the real data.

Because filter selection was to be a one-time off-line process, it was possible to perform it using a relatively computationally inefficient language on inexpensive desktop personal computers. Optimization and evaluation software was programmed using Macintosh Common LISP version 2.0.1 (Apple Computer) on Apple Macintosh 68030 (IIci and IIfx models) and 68040 (Quadra 950 and IIci with Radius Rocket accelerator) computers. Common LISP was chosen because of the ease with which code could be developed and modified, and the availability of implementations of the genetic algorithm and genetic programming in Common LISP. In fact, at the time that this study was started, genetic programming had not been implemented in any other language.

The genetic algorithm was employed in the GAL software of Spears (1991), ported to Macintosh Common LISP for this project. This uses Baker's SUS selection method, and was run with a chromosome string of length 56 bits, encoding seven 8-bit words to represent the tap weights of an FIR filter of window width seven. Both plain and gray-scale encoding were used in separate runs. Production runs consisted of populations of size 5000 with a mutation rate of 0.001 and a one-point crossover rate of 0.6. Prior development runs assessed the appropriateness of these settings, and demonstrated that increasing the mutation rate and reducing the crossover rate (making the search more random and less evolutionary) reduced the stabilization of fitness and lessened the best fitnesses attained. Production runs were terminated when the best fitness had long stabilized.

Genetic programming was performed using the Simple LISP implementation of Koza (1992), incorporating performance enhancements which were specific to the version of Macintosh Common LISP which was being used. These accelerated the evaluation of S-expressions within the population without any loss of accuracy. Exploratory series were undertaken initially to investigate the optimal settings for values within Koza's 'tableau', after which there was a production series in which groups of two to ten runs took several days or weeks to complete on each occasion.

Genetic programming used the input data values and generated random real numbers as the terminal set, and the four real arithmetic operators (`+`, `-`, `*`, and division protected from divide by zero errors) as the function set. It was thus configured to optimize FIR filters of window width seven. Initial populations of 500–2000 S-expressions were generated using Koza's ramped half-and-half method, with a maximum depth

of six. The selection method employed was a standard fitness proportionate method with reproduction fraction (the proportion of the population selected for reproduction) 0.1 and a maximum depth after crossover of 17. Each run was performed for 51 or 101 generations, but none terminated because the number of 'hits' or fitness was high enough to meet predetermined criteria.

The optimization of stack filters was very similar, using standard threshold decomposed data points (as described above) over the window of width seven as the terminal set. The function set consisted of logical NOT, AND, and OR operations, and other parameters were the same as used for FIR filters. The first two stages in stack filtering are inefficient when implemented in Common LISP by S-expressions, and runs commonly took several days to complete. Further details of the settings used are given by Oakley (1994a). No attempt was made to use the more recent technique of automatic function definition (Koza 1994).

G3.1.4 Results

The fittest filter discovered in the whole project was the fittest found by genetic programming on stack filters. This can conveniently be seen as a median filter of window width three coupled with a fragment of a median filter of window width seven, and is best described by the following LISP S-expression:

```
(OR (AND Y3 Y4) (AND Y3 Y5) (AND Y4 Y5) (AND Y1 Y4 Y7))
```

where the values within the window are from `Y1` to `Y7`. This had a fitness of 0.0816, which corresponds to an rms error of 11.2, that is, 6% of the average input data value. Application of this filter to an abundance of real-world data has confirmed that it achieves the required attenuation of noise without significant distortion of the underlying signal.

Significantly less fit, but second and third respectively, were the heuristically found median 5 and median 3 filters (fitnesses 0.0787 and 0.0776). Following these was a filter found by seeding the initial population of a genetic programming run with the fittest filter found by application of the genetic algorithm, a simple 7-tap FIR filter (fitness 0.0695). The next was that filter resulting from the genetic algorithm (fitness 0.0685), and a bell-shaped FIR filter of equal fitness suggested by an expert human advisor. The gray-scale implementation of the genetic algorithm did not perform quite as well as that using plain encoding, returning the next fittest filter (fitness 0.0684).

The fittest FIR filter resulting from genetic programming alone was 12th fittest overall (fitness 0.0489), ahead of the best of the RC filters, that with a time constant of 0.0375 seconds (0.0459). This was not much better than the fitness resulting from not using a filter at all (0.0437), which was in turn much better than the best filter built into the blood flow measurement system, an RC filter with time constant 0.1 seconds and a fitness of only 0.0266.

Preliminary indications are that the best stack filter, found by genetic programming, has qualities which make it superior to many heuristically chosen filters in other real-world applications, and it may prove to be a design which merits entry into the signal processing repertoire. Although not ideal for implementation in software, it has now been used for the off-line processing of many millions of data points. Most recently, it has been reimplemented in compiled C++ code and incorporated into a real-time data processing system for the blood flow measurement hardware. This is in daily use as a research and clinical medical tool, comfortably outperforming standard blood flow measurement systems of the same kind (Oakley 1994b).

G3.1.5 Conclusions

Genetic programming has proved to be a very effective way of developing an optimized filter to reduce noise in this situation. The fittest filter produced by genetic programming outperformed the best suggested by experts, or derived from optimizations using the genetic algorithm. Although some of this may be attributable to the fact that genetic programming was able to optimize a class of filter—stack filters—which appeared to perform well in this particular application, it was also able to improve upon the performance of the fittest FIR filter produced by the genetic algorithm.

It is important to recognize that genetic programming has a number of advantages over other optimization methods in this type of problem. Because it is relatively easy to frame an optimization problem in terms which are amenable to genetic programming techniques, this technique can be applied quickly to a much wider range of problems than other evolutionary methods, and by practitioners who have neither the resources nor the desire to develop a representation of the problem in terms of the genetic

algorithm. Once the technique is understood, modifications can be made to set the Simple LISP code up to optimize a class of filter in minutes rather than days or weeks.

The main drawback of genetic programming, prolonged computational runtime, can be ameliorated by using more expensive and thus higher-performance computer resources and more efficiently compiled languages, although even they are unlikely to make its use practical in on-line adaptive signal processing.

References

Ansari N, Huang Y and Lin J-H 1992 Adaptive stack filtering by LMS and perceptron learning *Dynamic, Genetic, and Chaotic Programming* ed B Soucek (New York: Wiley–Interscience) pp 119–43

Chu C-H H 1989 A genetic algorithm approach to the configuration of stack filters *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, June 1989)* ed J D Shaffer (San Mateo, CA: Morgan Kaufmann) pp 218–24

DeFatta D J, Lucas J G and Hodgkiss W S 1988 *Digital Signal Processing* (New York: Wiley)

Etter D M, Hicks M J and Cho K H 1982 Recursive adaptive filter design using genetic algorithms *Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing* vol 2 (Piscataway, NJ: IEEE) pp 635–8

Koza J R 1992 *Genetic Programming. On the Programming of Computers by Means of Natural Selection* (Cambridge, MA: MIT Press)

——1994 *Genetic Programming II: Automatic Discovery of Reusable Programs* (Cambridge, MA: MIT Press)

Oakley E H N 1994a Two scientific applications of genetic programming: stack filter and non-linear equation fitting to chaotic data *Advances in Genetic Programming* ed K E Kinnear (Cambridge, MA: MIT Press) pp 369–89

——1994b Techniques for filtering noise from laser Doppler rheometer data *Progress in Microcirculation Research* ed H Niimi, M Oda, T Sawada and R-J Xiu (Oxford: Pergamon) pp 493–6

Spears W M 1991 *GAL* Common LISP source code published electronically at ftp.aic.nrl.navy.mil

Wendt P D, Coyle E J and Gallagher N C 1986 Stack filters *IEEE Trans. Acoust. Speech Sig. Process.* **ASSP-34** 898–911

Widrow B and Stearns S D 1985 *Adaptive Signal Processing* (Englewood Cliffs, NJ: Prentice-Hall)

Further reading

1. Chu C-H H 1989 A genetic algorithm approach to the configuration of stack filters *Proc. 3rd Int. Conf. on Genetic Algorithms* ed J D Shaffer (San Mateo, CA: Morgan Kaufmann) pp 218–24

   An elegant and successful technique for optimizing stack filters using the genetic algorithm.

2. Koza J R 1992 *Genetic Programming. On the Programming of Computers by Means of Natural Selection* (Cambridge, MA: MIT Press)

   The standard work on genetic programming. Comprehensive and exhaustive.

3. Koza J R 1994 *Genetic Programming II: Automatic Discovery of Reusable Programs* (Cambridge, MA: MIT Press)

   Introduces the important technique of automatically defined functions (ADFs), as well as containing an excellent annotated bibliography of early work on genetic programming.

G3.2 Genetic algorithms for the optimization of combustion in multiple-burner furnaces and boiler plants

Terence C Fogarty

Abstract

The problem of automating the production of a rule-based system for optimizing combustion in multiple-burner furnaces and boiler plants is presented. A solution using genetic algorithms to learn individual rules in a classifier system is described. Results of experiments using this system on simulations of multiple-burner installations are presented. Finally the limitations of the approach are discussed.

G3.2.1 The problem

Fogarty (1988) developed, tested, and successfully used a rule-based system for optimizing combustion in multiple-burner installations on a 12-burner zone of the 108-burner furnace of a continuous annealing line for rolled steel. The idea was to develop a rule-based system for optimizing combustion in any multiple-burner installation as shown in figure G3.2.1. Here, a state encoder classifies the oxygen (ox) and carbon monoxide (co) readings for input to the rule-based system while an action decoder translates the output of the rule-based system into movements of the air inlet valves. A cost function is calculated on the basis of the oxygen, carbon monoxide, and temperature (T) readings.

When demonstrating the system on a double-burner boiler in the steam generating plant of a tinplate finishing works it was found that the rulebase elicited from the experts was not very efficient at dealing with the modulation of firing levels in response to changing levels of demand from the works. The rulebase was not built with this problem in mind since the multiple-burner furnace on which it was developed only had one firing level and zones were turned on or off to satisfy requirements rather than having their firing levels modulated. The rules had to be modified manually to work successfully (Fogarty 1989). The resulting rulebase is shown in figure G3.2.2. The problem is to automate the process of building sets of rules for optimizing combustion in multiple-burner furnaces and boiler plants.

G3.2.2 Method of solution

G3.2.2.1 Background

Holland (1975) encapsulated, in the *genetic algorithm*, the process of the evolution of a natural system to provide us with a method for specifying an artificial system which can adapt to a given environment. While Smith (1980) used the genetic algorithm to optimize complete systems specified as sets of rules, the *Pitt* approach developed at the University of Pittsburgh (De Jong 1988), Holland developed the *Michigan* approach of using the genetic algorithm to discover individual rules within a rule-based system (Holland and Reitman 1978). Whatever approach is used, the resulting rule-based systems are known as classifier systems and the individual rules they contain as classifiers. For a fuller discussion of classifier systems see Section B1.5.2 of this handbook.

B1.2

B1.5.2

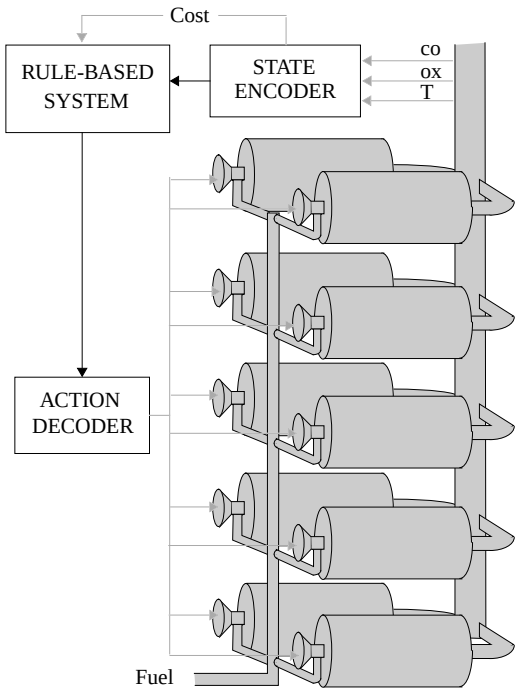


Figure G3.2.1. Multiple-burner combustion control.

Booker (1985) introduced the mechanism of restricted mating to focus the operation of the genetic algorithm in a classifier system on subpopulations of similar rules. Wilson (1987) showed how classifier systems could deal with immediate reinforcement by introducing the idea of the match set in which all classifiers activated by the current input have their weights updated according to immediate reward.

oxygen v. high	Reduce air to all burners by 4%	Reduce air to all burners by 4%	Lean burner correction routine Incs = 4%	Rich/lean correction routine Incs = 4%
oxygen high	Reduce air to all burners Incs = 1%	Lean burner correction routine Incs = 2%	Rich/lean correction routine Incs = 2%	Rich burner correction routine Incs = 4%
oxygen O.K.	Lean burner correction routine Incs = 1%	Rich/lean correction routine Incs = 1%	Rich burner correction routine Incs = 2%	Increase air to all burners by 4%
oxygen low	Do nothing	Rich burner correction routine Incs = 1%	Increase air to all burners Incs = 2%	Increase air to all burners by 4%
	carbon monoxide low	carbon monoxide O.K.	carbon monoxide high	carbon monoxide v. high

Figure G3.2.2. The rulebase for the double-burner boiler.

G3.2.2.2 Approach adopted

The approach adopted here is based on the *Michigan* style classifier system (Holland 1986). Building on the contribution of Wilson and Booker we restrict the operation of the genetic algorithm to the match set which provides an elegant mechanism for identifying subpopulations of similar classifiers.

The simple classifier system developed, shown in figure G3.2.3, consists of a set of fully specific condition/action rules each of which has an associated weight, representing its cost or value, and time, recording when it was created or produced. The operation of the classifier system is shown in figure G3.2.4. To begin, the classifier system contains no rules. Messages are received from the environment and all rules, if any, with conditions that match the environmental messages are activated. If the number of activated rules is below a certain size, let us say P, a new rule is created using the cover operator (Wilson 1985). This has conditions that match the environmental messages and an action randomly chosen from the set of allowable actions.

If the number of activated rules is equal to P, a new rule is produced from the set of activated rules with the genetic algorithm. One rule is chosen and copied as the basis for a new rule, using the weights

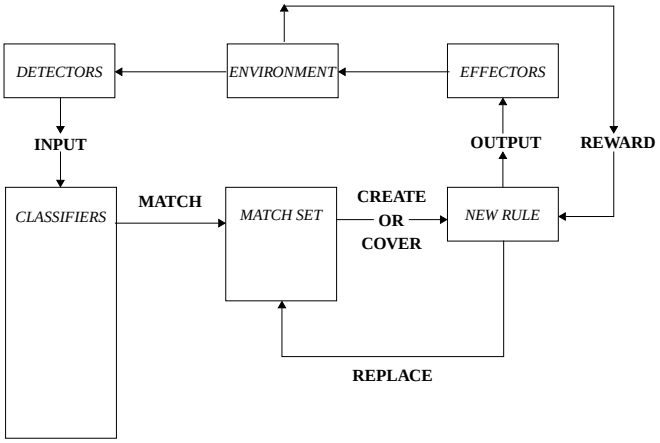


Figure G3.2.3. The components of the classifier system and their interaction.

```
Set the maximum size of the match set to P
Create an empty array as the main classifier store
Create an empty array for the match set
Repeat
    Read inputs from the environment
    Move all classifiers with conditions matching the inputs to the match set
    If the size of the match set is less than P
        Create a new classifier with conditions equal to the inputs and a random action
    Else
        Use the genetic algorithm on the population of the match set to create a new classifier
    Let the action of the new classifier be the system's output
    Collect reward (or punishment) and assign as the weight of the new classifier
    Replace the oldest classifier in the match set with the new classifier
    Move all the classifiers in the match set back into the main classifier store
Until end
```

Figure G3.2.4. Description of the operation of the classifier system.

of the activated rules as a probability distribution for the purpose of selection. With a high probability a second rule is chosen, using the same method, and a randomly selected part of it is copied over the corresponding part of the copy of the first rule. The action of the resulting single new rule is mutated at a randomly generated point with a low probability to produce a new rule. The new rule, whether created with the cover operator or produced with the genetic algorithm, posts its action to the message list and this becomes the output of the system. The reward or punishment that is received from the environment becomes the weight of the new rule—the weights of all existing rules in the classifier system remain unaltered.

If the new rule was created with the cover operator it now becomes part of the classifier system. If it was produced with the genetic algorithm the oldest activated rule is selected for deletion and this is immediately replaced with the new rule. Deletion based on weight has also been used. Finally new messages from the environment are read and the process is repeated.

G3.2.2.3 Relation to other systems

The main difference between the simple system described above and more complicated versions of the classifier system is that the genetic algorithm is explicitly used for reinforcement as well as discovery. A new rule is created or produced at each interaction with the environment and its weight reflects the cost or value of its action to the system in the state defined by its conditions. Old rules are replaced when necessary so that the population of rules for a given set of conditions adapts to produce the best action for the indicated state of the environment. Actions that benefit the system in that state come to dominate the population with that set of conditions while those which do not wither away. Thus, a beneficial action for a given state can be randomly selected or discovered using crossover and mutation and then reinforced using selection. The value or cost of an action in a given state is given by the average strength of the corresponding rules while the probability of that action being taken in a given state is approximated by the sum of the strength of the corresponding rules relative to the sum of the strengths of rules with other actions for that state.

The genetic algorithm is based largely on that outlined by Holland and Reitman (1978) except in its use. They generated two parent rules from the set controlling the same effector based on their predicted payoff and crossed them to produce a new rule which replaced one of the oldest in the population as a whole. The system described takes the restricted mating of Booker (1985) to the extreme; rules with the same conditions form a distinct predefined species within the total population of the classifier system that adapts to give the best action for a given state.

G3.2.3 Experiments

A system based on the classifier system described above has been used to control simulations (Fogarty 1990) of multiple-burner installations and provides a good example of its operation. The inputs to the system are oxygen and carbon monoxide readings taken in the common flue of the burners, each classified as ‘very low’, ‘low’, ‘o.k.’, or ‘high’, giving 16 possible states in the environment. The system has fixed actions when either of the readings is very high. There are 96 possible outputs from the system to the air inlet valves controlling the air–fuel ratios of the burners. These are composed of the six qualitatively different actions applied by any of 16 different amounts. The qualitatively different actions are lean burner correction, rich burner correction, lean/rich burner correction, reduce air to all burners, increase air to all burners, and no action. These are detailed in figure G3.2.5. The first three of these are only applied to the current burner and attention is then switched to the next burner ready for the next action; the rest are applied to all of the burners. The cost of an action is the energy loss calculated from the oxygen and carbon monoxide readings together with the temperature (Fogarty 1991) in the common flue after that action has been performed.

The maximum number of activated rules P was 30 with the probability of a second rule being chosen for single-point crossover of 0.95 and a probability of mutation of 0.01. The rules are encoded with strings representing the conditions such as ‘low, high’ when the oxygen reading is low and the carbon monoxide reading is high and bits representing the qualitatively different actions together with their associated amounts. The first three bits of the action are used for the qualitative part of the action with a bias of three different representations for doing nothing as shown in table G3.2.1 and the other bits are a binary encoding of the amount.

```
LEAN BURNER CORRECTION ROUTINE:
    reduce air to the current burner
    IF CO increases significantly
        THEN increase air to the current burner
    ELSE leave
    move to next burner

RICH BURNER CORRECTION ROUTINE:
    increase air to the current burner
    IF CO is reduced
        THEN leave
    ELSE reduce air to the current burner
    move to next burner

RICH/LEAN CORRECTION ROUTINE:
    increase air to the current burner
    IF CO is reduced
        THEN leave
    ELSE reduce air to the current burner
            reduce air to the current burner
    IF CO increases significantly
        THEN increase air to the current burner
    ELSE leave
    move to next burner
```

Figure G3.2.5. Details of the actions.

Table G3.2.1. Coding for actions.

Code	Action
000	No action
001	No action
110	No action
011	Lean burner correction routine
101	Rich burner correction routine
111	Rich/lean burner correction routine
010	Reduce air to all burners
100	Increase air to all burners

G3.2.4 Results

The system has been run on ten different simulations of ten burner installations with two firing levels for 20 000 interactions each. In situations where the carbon monoxide reading is high there is a definite convergence of rules to the action of increasing air to all burners by about 10%. In situations where the carbon monoxide reading is very low there is a convergence of rules towards rich/lean or lean correction by various amounts or do nothing, depending upon the oxygen reading. The rulebase resulting from a run on one simulation is shown in figure G3.2.6. The system was run on a real multiple-burner installation but it has some obvious limitations.

oxygen v. high	Rich/lean correction routine Incs = 12%			
oxygen high	Rich/lean correction routine Incs = 12%			
oxygen O.K.	Do nothing			Increase air to all burners by 9%
oxygen low	Lean burner correction routine Incs = 7%			Increase air to all burners by 11%
	carbon monoxide low	carbon monoxide O.K.	carbon monoxide high	carbon monoxide v. high

Figure G3.2.6. Learned classifiers for the multiple-burner simulation.

G3.2.5 Discussion

The first observation to make is that a definite action is not learned for every situation encountered. Some situations are not entered enough times in a run so that the action taken in that situation is still random at the end of the run. Other situations have populations suggesting two different actions at the end of a run which may be due to the deselection strategy of scaled proportional replacement of the worst rather than the oldest rule but is more likely to be due to the fact that there are hidden states not identified by the system. The main reason is that the state space has to be discretized by an expert rather than learned by the system and an approach using point-based classifiers with nearest-neighbor matching has been proposed to overcome this (Fogarty and Huang 1992).

Secondly, the system can never learn an action that is qualitatively different from the ones it has at its disposal because the actions are sophisticated routines specified by an expert. They could be broken down into simpler actions from which new routines could be constructed but these would need to be able to make use of delayed rather than immediate reinforcement since some of them involve incurring a temporary loss in order to make a longer-term gain.

References

Booker L B 1985 *Intelligent Behaviour as an Adaptation to the Task Environment* PhD Dissertation, University of Michigan

De Jong K 1988 Learning with genetic algorithms: an overview *Machine Learning* **3** 121–37

Fogarty T C 1988 Rule-based optimisation of combustion in multiple burner furnaces and boiler plants *Eng. Appl. Artificial Intell.* **1** 203–9

——1989 Adapting a rule-base for optimising combustion on a double burner boiler *2nd Int. Conf. on Software Engineering for Real Time Systems (IEE Conf. Publ. 309)* p 106–10

——1990 Simulating multiple burner combustion for rule-based control *Syst. Sci.* **16** 23–38

——1991 Putting energy efficiency into the control loop *IMechE Technology Transfer in Energy Efficiency Session of Eurotech Direct 91* p 39–41

Fogarty T C and Huang R 1992 Systems control with the genetic algorithm and nearest neighbour classification *CC-AI* **9** 225–36

Holland J H 1975 *Adaptation in Natural and Artificial Systems* (Ann Arbor, MI: University of Michigan Press)

——1986 Escaping brittleness: the possibilities of general-purpose learning algorithms applied to parallel rule-based systems *Machine Learning, an Artificial Intelligence Approach* vol II, ed R S Michalski, J G Carbonell and T M Mitchel (Los Altos, CA: Morgan Kaufmann)

Holland J H and Reitman J S 1978 Cognitive systems based on adaptive algorithms *Pattern-directed Inference Systems* ed D A Waterman and F Hayes-Roth (New York: Academic)

Smith S 1980 *A Learning System Based on Genetic Algorithms* PhD Dissertation, University of Pittsburgh

Wilson S W 1985 Knowledge growth in an artificial animal *Proc. 1st Int. Conf. on Genetic Algorithms (Pittsburgh, PA, July 1985)* ed J J Grefenstette (Hillsdale, NJ: Lawrence Erlbaum) pp 16–23

——1987 Classifier systems and the animat problem *Machine Learning* **2** 199–228

G3.3 An application of genetic algorithms to air combat maneuvering

Robert E Smith and Bruce A Dike

Abstract

This case study reports on a project where a genetics-based machine learning system acquired rules for novel fighter combat maneuvers through simulation. In this project, a genetic learning system (GLS) was implemented to generate high-angle-of-attack air combat tactics for the NASA X-31 research aircraft. The GLS, which was based on a learning classifier system (LCS) approach, employed a digital simulation model of one-versus-one air combat and a genetic algorithm to develop effective tactics for the unconventional flight modes of the X-31. The resulting maneuvers allowed the X-31 to successfully exploit its poststall capabilities against a conventional fighter opponent, demonstrating the ability of the GLS to discover novel tactics in a dynamic air combat environment. Moreover, the project demonstrates how genetic machine learning can acquire rules that implement novel approaches to unforeseen problems via experience with simulations.

G3.3.1 Project overview

This case study reports the results of a joint project conducted by McDonnell Douglas Aerospace (MDA), The University of Alabama, and NASA to investigate the acquistion of rules for novel combat maneuvers for highly agile fighter aircraft via genetics-based machine learning in dogfight simulations. The benefits of this approach are that an automated system can explore the advantages of new aerodynamic characteristics of a fighter, without flying a prototype in costly test flights, or the expense of test pilots interacting in simulations. It is important to distinguish this work from studies where the ultimate goal is a set of rules to automatically control a plane or some other system. In this study, the goal was simply the discovery of rules that implement novel maneuvers. This was accomplished through the genetic learning system's experience with simulated combat. After the completion of learning, the rules convey the knowledge acquired in a straightforward way that can be evaluated and possibly used by actual pilots. Moreover, rules acquired in this manner could also be used to supplement a knowledge base of combat maneuvers for other studies.

The remainder of this section will discuss project objectives and genetics-based machine learning techniques, experimental results, and their evaluation. Final comments suggest directions for further research.

G3.3.1.1 Project objectives

The primary objective of the project was to provide a method using computer algorithms to develop optimal air combat tactics for the X-31 aircraft. The tactics to be developed were for the within-visual-range (WVR) close-in combat (CIC) high-angle-of-attack arena. The engagement simulation was required to provide a means to input an aircraft model, including realistic constraints and inputs for the control system and weapon system. The main weapon focus was on high-velocity gun employment, with little emphasis on missile systems. Tactic optimization was to be based on standard single-engagement measures

of effectiveness, such as time on advantage and time to first kill. The algorithm was required to allow for easy changes in aircraft characteristics, such as thrust-to-weight ratio, and to permit sensitivity studies for these characteristics.

G3.3.1.2 Tactic optimization problem

The range of possible maneuvers available to poststall tactic (PST) aircraft vastly exceeds that of conventional aircraft (Doane *et al* 1989). PST aircraft are not constrained to standard flight envelopes or flight attitudes. To fully address the question of PST effectiveness, a broad investigation of agility in CIC must be conducted without involving exhaustive experimentation in manned simulations or flight test ranges. These facilities provide the most realistic environments for controlled testing of fighter effectiveness. However, manned testing requires trial-and-error tactic development by highly trained pilots, and the cost of comprehensive tactical analysis is prohibitive. Man-in-the-loop simulation or flight testing can explore at most a small subset of meaningful PST maneuver options, and only at great cost and effort.

Adequate exploration of PST maneuver possibilities requires some form of automated search and optimization process. Off-line methods are needed to define and evaluate optimal PST tactics efficiently and systematically. If the best or near-best maneuvers are not identified, the full potential and value of PST aircraft may not be realized.

G3.3.1.3 Shortcomings of conventional trajectory optimization methods

Off-line methods for trajectory optimization have been employed for PST research for a number of years. The standard approaches usually involve one- or two-sided optimal control algorithms to solve a differential game problem. While precise and well understood, these calculus-based methods have significant shortcomings, which fall into three categories.

First, the methods are essentially local in nature, since they either implicitly or explicitly employ derivatives or gradient searches to locate the solution. The second difficulty with conventional methods is the large analytical and computational burden generated by the solution techniques. The third and primary shortcoming with conventional trajectory optimization methods is the gross oversimplification of the many operational factors present in air combat.

G3.3.2 Technical approach

This project pursued a solution strategy for X-31 tactic optimization using *genetic algorithms* (GAs) and machine learning which addresses the shortcomings of current methods. This section describes the specific approach for accomplishing the X-31 tactic study objectives. B1.2

G3.3.2.1 Genetic learning system (GLS) approach

The genetic learning system (GLS), developed by MDA and the University of Alabama, was used as the tactic optimization procedure for this project. Based on a stimulus–response *learning classifier system* (LCS) approach (Holland *et al* 1986), the GLS treated both the maneuvers and the trigger conditions as unknown quantities to be optimized simultaneously. The GLS determined which maneuvers to perform, and when to perform them. The output of the GLS was a set of IF–THEN statements which specified aircraft control commands as a function of flight condition and relative engagement geometry, similar to a set of closed-loop control laws. The IF–THEN statements, or classifiers, were tested at each time step in the engagement to determine which should be activated. At the end of the engagement, the effectiveness score was provided to the GLS for processing the next generation of classifiers. B1.5.2

A GA produced a population of IF–THEN rules, which were input into the Advanced Air-To-Air System Performance Evaluation Model (AASPEM, a digital computer simulation of air combat) simulation to control the X-31 aircraft. The opponent aircraft was a modern fighter controlled by the standard CIC tactic logic. An air combat engagement was simulated, and the results were used as fitness values by the GA to produce the next generation of rules. The process was repeated until the engagement scores exhibited no further improvements for the X-31. The tactics from the best case in the run sequence were extracted and processed for further analysis.

G3.3.2.2 Genetic learning system architecture

Figure G3.3.1 illustrates the GLS architecture. The system was composed of six elements: (i) a GA, which produces IF–THEN classifier (rule) populations; (ii) an environment simulation (AASPEM) which exercises the rule population and produces an engagement measure of merit for the rule population; (iii) an environment interface (EI), which determined the classifiers to activate within the engagement simulation; (iv) a credit allocation algorithm, which converted the engagement measure of merit into fitness values for the GA; (v) a setup procedure which randomly initialized the population for the first generation, and (vi) a GLS executive, which controlled the process.

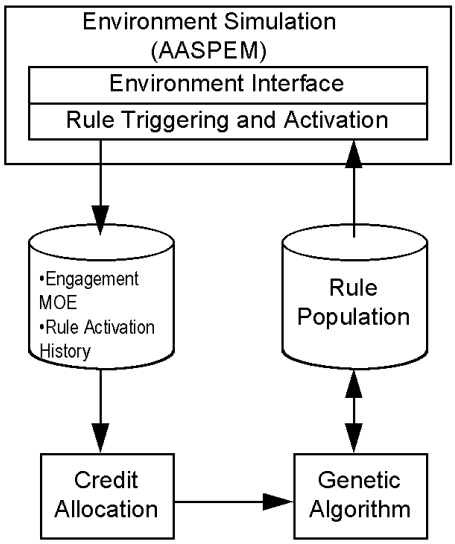


Figure G3.3.1. The GLS architecture.

(i) *Genetic algorithm.* The classifier population for each generation was generated by a conventional GA, using the fitness values assigned by the allocation of credit function. The GA employed *tournament selection*, *single-point crossover*, and a nominal *mutation rate* of one per thousand. In this implementation, each classifier was 28 characters long, with 20 left-hand-side (LHS) characters and eight right-hand-side (RHS) characters. The LHS alphabet consisted of $\{0, 1, \#\}$, and the RHS alphabet consisted of $\{0, 1\}$. Initial runs using a population size of 50 rules showed efficient performance, but a larger population of 200 rules was later used to provide greater protection against proliferation of nonfiring *parasite* rules. C2.3 C3.3, C3.2

(ii) *Environment simulation.* AASPEM was used to simulate a single air-to-air engagement with each rule population. Each iteration cycle, or generation, a single one-versus-one air combat engagement was generated by AASPEM. The X-31 employed the current population of classifiers (rules) developed by the GLS for its tactics. The opponent aircraft used the conventional CIC maneuvers encoded in AASPEM.

(iii) *Environment interface.* At each time step (0.10 s) in the simulation, the environment interface determined whether any of the classifiers in the population were triggered. If more than one classifier was triggered at the same time, the EI activated the classifier with the highest residual fitness value. The triggering–activation process continued until the engagement was terminated. At the end of the run, the final engagement score was calculated, and all activated classifiers were identified.

(iv) *Allocation of credit.* The allocation of credit function used the AASPEM engagement score and activation list generated by the EI to assign fitness values for each individual classifier. The allocation of credit function also ensured that information from previous generations was included in the selection process for the next generation.

Four steps were involved in the allocation of credit function. In the first step, all activated classifiers were directly assigned the score of the last completed engagement. The score (figure G3.3.2) was based on average angular advantage (opponent target aspect angle minus own-ship target aspect angle). To encourage maneuvers which might enable gun firing opportunities, an additional score was added when the target was within 5 degrees of the aircraft's nose. This epochal method ensured that each activated classifier was equally rewarded on the basis of the final engagement score.

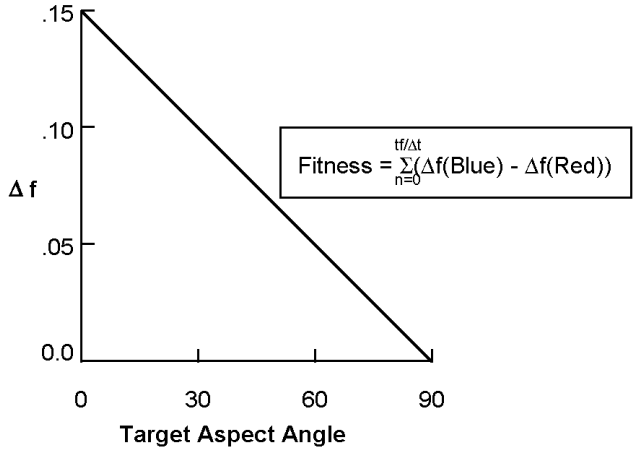


Figure G3.3.2. The X-31 GLS fitness function.

In the second step, nonactivated classifiers were assigned the average fitness value of their parents from the previous generation. This inheritance process insured that elements of high-performance classifiers were preserved across generations even if they were not always activated.

In the third step, a tax was applied to nonfiring classifiers to discourage the proliferation of parasite classifiers which contain elements of high-performance classifiers but have insufficient material for activation.

In the final step, all nonfiring classifiers which were identical to a firing classifier were reassigned the firing classifier's fitness. This step reduced the classifier fitness noise level.

This allocation of credit process accomplished the dual objectives of reinforcing the activated classifiers while retaining previously effective classifier elements through the inheritance mechanism.

G3.3.2.3 Genetic learning system search space

The GLS search space was composed of two components, an engagement space, and a maneuver set. The engagement space was partitioned into discrete cells so that, at any time in the engagement, the flight states of each aircraft and their relative geometries could be associated with a specific trigger condition. The # character in the $\{0, 1, \#\}$ syntax effectively created OR cells in the engagement condition matrix. The X-31 GLS engagement space partitioning is shown in figure G3.3.3. Each row in the table represents a range of values in a classifier's condition or action. For instance, the first row represents eight possible ranges of own-ship aspect angle, which account for three bits of the classifier condition. Therefore, the first nine rows give the value ranges for all 20 bits of the classifier conditions, and the remaining three rows give the eight bits of the classifier actions. Other partitionings are possible with straightforward changes to the coding convention of the state variables.

The maneuver set was defined by own-ship bank angle (relative to the opponent maneuver plane), angle of attack, and throttle setting. These controls, when applied in 0.1 s time increments, were sufficient to perform any desired flight maneuver. AASPEM automatically limited the g levels of the maneuvers to values achievable by the aircraft and pilot. Manually sifting through search space was impractical; over 10^{11} combinations of conditions and maneuvers are possible in the example. The GLS identified the most effective combinations of engagement conditions and maneuver commands.

G3.3.3 X-31 tactic optimization results

G3.3.3.1 X-31 tactic development process

In the tactic generation phase, the GLS–X-31 maneuver generation procedure involved three steps: (i) case selection; (ii) GLS processing; and (iii) maneuver analysis. The GLS processing was performed with different cases until the test matrix was filled. The results of each case were stored for additional review and analysis by project engineers and test pilots.

Conditions	Ownship Aspect (deg)	< 45	45-90	90-135	135-180	180-215	215-270	270-315	315-360
	Opponent Aspect (deg)	< 45	45-90	90-135	135-180	180-215	215-270	270-315	315-360
	Range (ft)	< 1000	1000-4500	4500-7500	> 7500				
	Airspeed (kts)	< 200	200-350	350-480	> 480				
	Delta Airspeed	< -50	-50 - 50	50-100	> 100				
	Altitude (ft)	< 10k	10k-20k	20k-30k	> 30k				
	Delta Altitude	< -2000	-2k - 2k	2k-4k	> 4k				
	Climb Angle (deg)	< -30	-30 -30	30-60	> 60				
	Opponent Climb Angle	< -30	-30 - 30	30-60	> 60				
Commands	Speed (kts)	100	200	350	480				
	Relative Bank Angle (deg)	0	30	45	90	180	-30	-45	-90
	Angle of Attack (deg)	0	10	20	30	40	50	60	70

Figure G3.3.3. GLS search space.

G3.3.3.2 The test matrix

Test matrix definition. A two-tier approach was employed to define run conditions and GLS parameters. First, a baseline matrix of starting positions, relative geometries, and energy states was identified in conjunction with NASA requirements. The primary source document for this step was the X-31 Project Pinball II Tactical Utility Summary, which contained results from manned simulation engagements conducted in 1993 at Ottobrunn, Germany. Initial findings from the X-31 Tactical Utility Flight Test conducted at Dryden Flight Research Center were also used to to compare with results from this project.

The baseline test matrix (figure G3.3.4) was based on X-31 manned simulation and flight test conditions, and was tailored to the X-31 performance envelope, flight test range constraints, and other testing considerations. The first four start conditions (defensive (DEF), offensive (OFF), slow-speed line abreast (SSLA), and high-speed line abreast (HSLA)) were derived directly from the Pinball II project. The fifth start condition (high-speed head-on pass (B2BH)) was added to the matrix to provide an additional

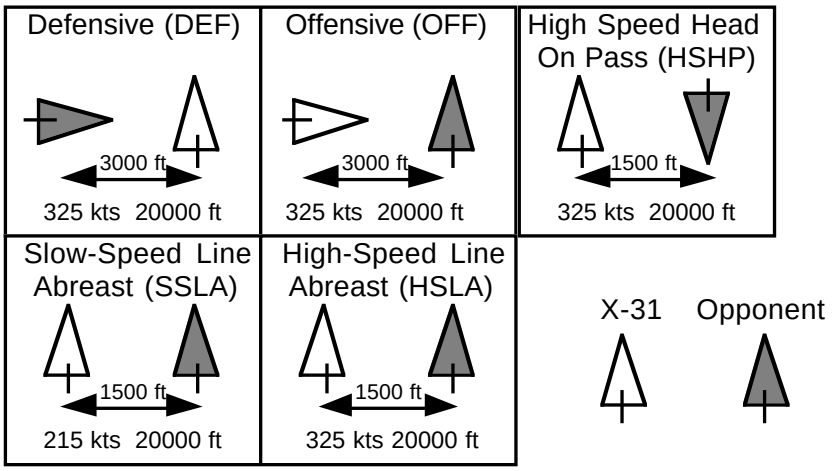


Figure G3.3.4. The baseline test matrix.

geometry which would not exclusively result in a close-turning fight. The opponent aircraft was an F/A-18. The baseline matrix formed a set of core conditions to generate X-31 tactic results for a balanced cross-section of tactically relevant conditions. The test conditions specified the initial geometries and X-31 and opponent speeds, altitudes, and ranges.

Sensitivity cases. Additional cases were evaluated with the GLS to explore the effects of various sensitivity factors on optimal tactic generation. The second tier of the test matrix was a series of experiments expanding the scope of the baseline matrix, which included the following.

(i) *150% thrust-to-weight (T/W).* The thrust level of the X-31 was increased by 50% across the flight envelope.

(ii) *90° alpha.* The maximum angle of attack of the X-31 was increased to 90° below 265 kts.

(iii) *150% T/W and 90° alpha.* Increased thrust-to-weight ratio and maximum alpha were combined.

(iv) *30° alpha.* The X-31 was limited to 30 degrees maximum angle of attack throughout the flight envelope. This case was included to establish a level of conventional performance for the X-31.

(v) *Opponent aircraft with 150% T/W.* The thrust level of the opponent aircraft was increased by 50% across the flight envelope. This case was added to examine the sensitivity to opponent performance levels.

Additional cases were defined to explore the effects of various factors on the X-31 tactics, such as aircraft performance and configuration changes, opponent maneuver strategies, alternate MOE, and GLS control parameters. These cases were not evaluated in all starting conditions, but were examined in specific scenarios to investigate their impact on GLS and X-31 performance.

G3.3.3.3 Genetic learning system tactic results

The GLS processing cycle was executed for each case in the test matrix. The total number of runs in a single processing cycle varied from 200 to 500. Other GA parameters are shown in table G3.3.1. The best run from each processing cycle was extracted for review and analysis. Each case is identified by the run number from the GLS processing sequence, the start condition identifier, and case description.

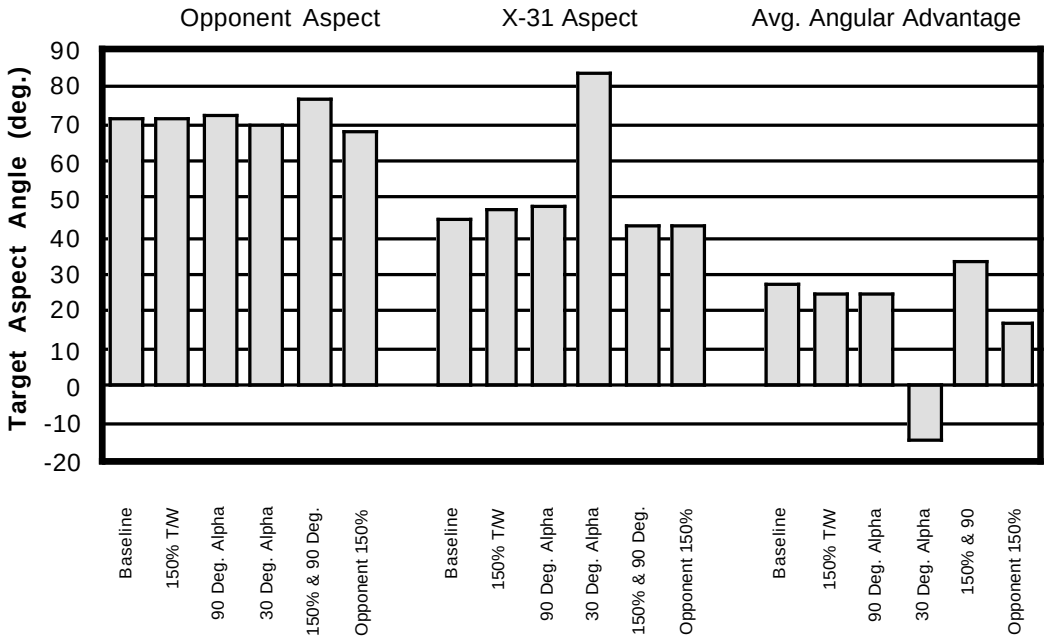


Figure G3.3.5. Average target aspect angle.

Table G3.3.1. GA parameters.

Population size	200
String length	28
P(crossover)	0.95
P(mutation)	0.01
Selection method	Tournament

Average angular advantage. Figure G3.3.5 shows the average target aspect angles (angle of the line-of-sight vector from own-ship to the target measured off own-ship's nose) for the best result of each case in the test matrix. The average angular advantage (difference between opponent aircraft aspect and X-31 aspect) is also shown in figure G3.3.5. The most pronounced difference occurred in the 30° alpha case with the X-31 aspect angle, where a noticeable lack of nose pointing ability was apparent. Except for the 30° alpha case, all other cases exhibited a positive advantage for the X-31 over the opponent, with the greatest advantages occurring in the 150% T/W and 90° alpha case. A score of zero indicates no cumulative angular advantage for either aircraft.

Average target aspect angles are shown for each starting condition of the baseline X-31 case in figure G3.3.6, and for the 150% T/W and 90° alpha X-31 case in figure G3.3.7.

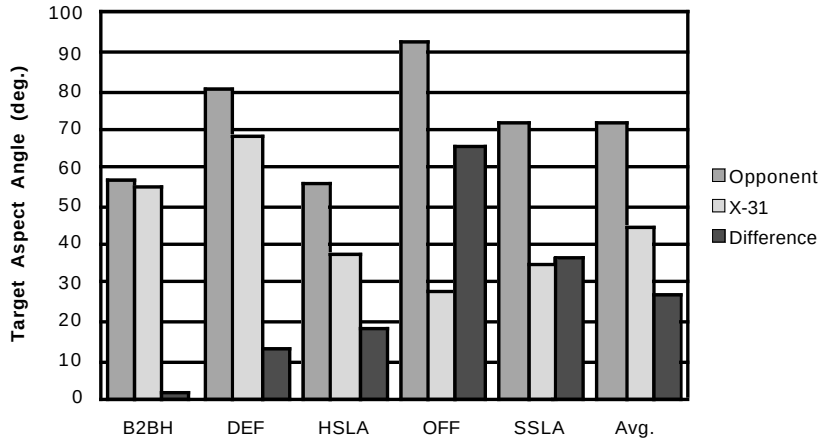


Figure G3.3.6. Average target aspect angles for the baseline case. In each case, the bars represent, from left to right, the angles for opponent, X-31, and the difference.

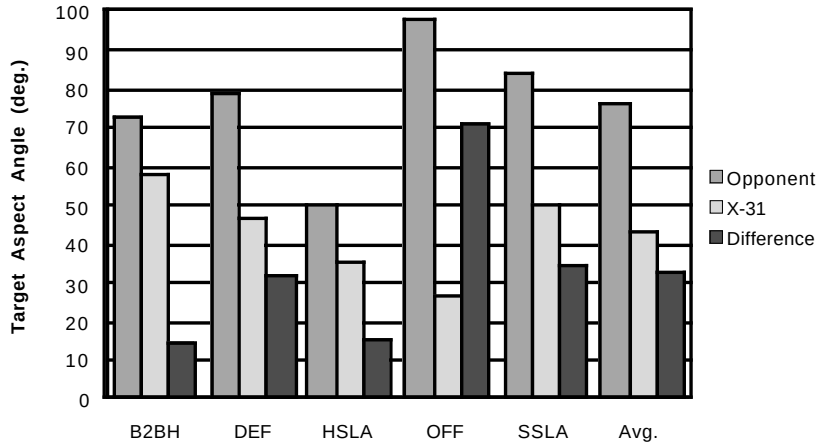


Figure G3.3.7. Average target aspect angles for the 150% T/W and 90° alpha case. In each case, the bars represent, from left to right, the angles for opponent, X-31, and the difference.

Target aspect angle time histories. Figures G3.3.8–G3.3.12 contain time history plots of target aspect angles for the best result of each case in the baseline matrix, illustrating the times of greatest and least advantage of the X-31 over the opponent.

The plots of engagements containing the best results for the baseline X-31 are illustrated in figures G3.3.13–G3.3.15. Side-view figures contain altitude lines to each aircraft icon. The X-31 trajectory is indicated by the smaller aircraft icon and the opponent aircraft trajectory is indicated by the larger icon.

Figure G3.3.13 illustrates the baseline X-31 HSLA case. The GLS commands the X-31 to execute a high alpha coning maneuver followed by a tight-radius 'helicopter gun attack' inside the radius of the opponent while maintaining a nose pointing attitude toward the target. Figure G3.3.14 illustrates an X-31 with 150% T/W and 90° alpha in the high-speed head-on pass engagement (B2BH) case. In this engagement, the GLS develops a Herbst-type maneuver with the X-31 climbing and rotating about the velocity vector, successfully reversing inside the opponent's turn radius. These examples illustrate the ability of the GLS to discover innovative maneuvers in different tactical situations by exploiting the maneuvering potential of the X-31.

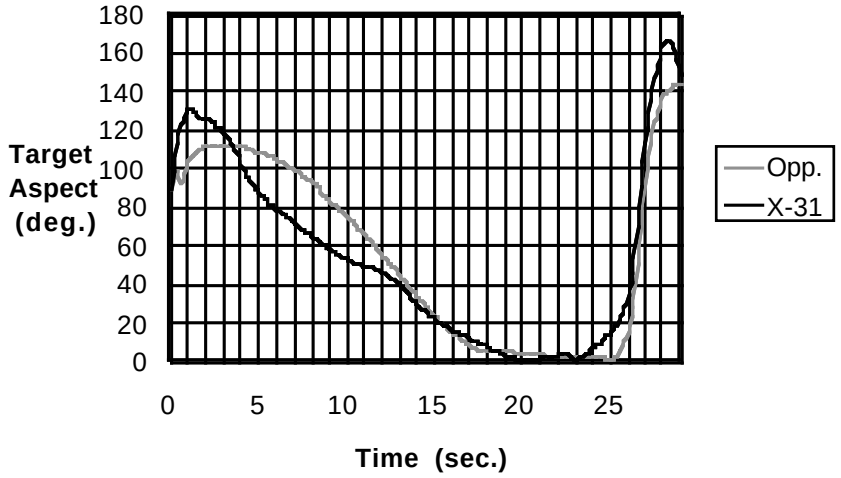


Figure G3.3.8. Baseline B2BH aspect angle history.

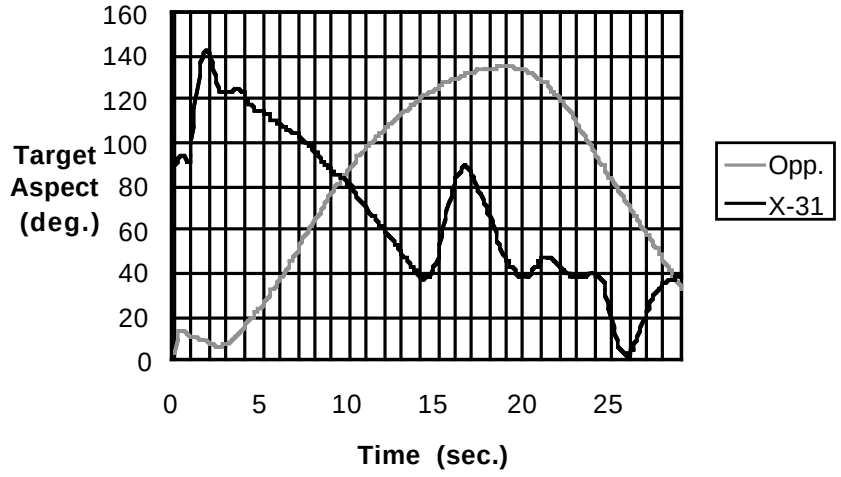


Figure G3.3.9. Baseline DEF aspect angle time history.

G3.3.4 Conclusions

The GLS provided a method for systematically generating a comprehensive rulebase for a wide set of tactical situations. These rules have received positive evaluations from actual test pilots (Smith and Dike

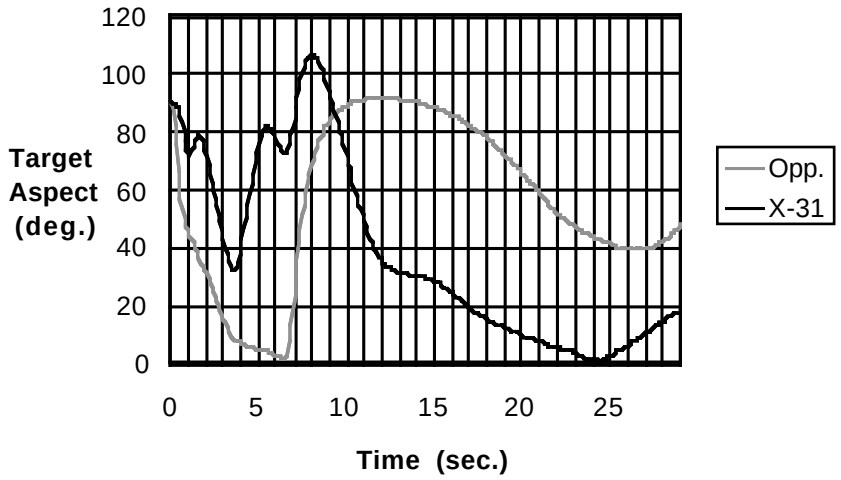


Figure G3.3.10. Baseline X-31 HSLA aspect angle time history.

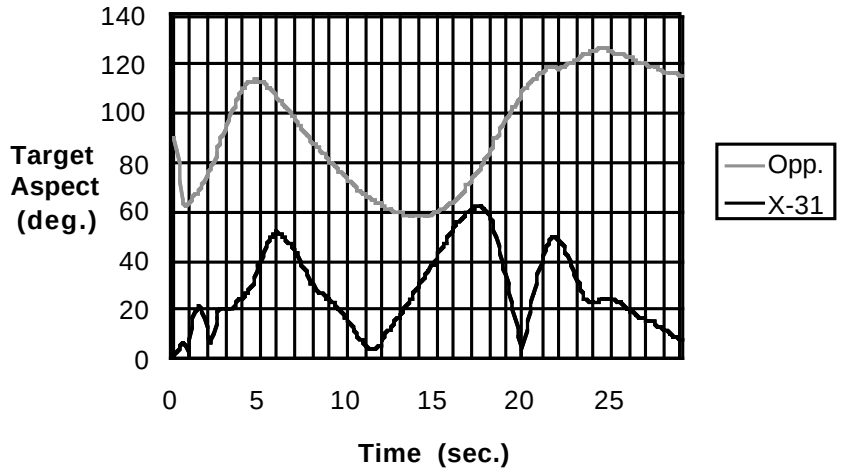


Figure G3.3.11. Baseline X-31 OFF aspect angle time history.

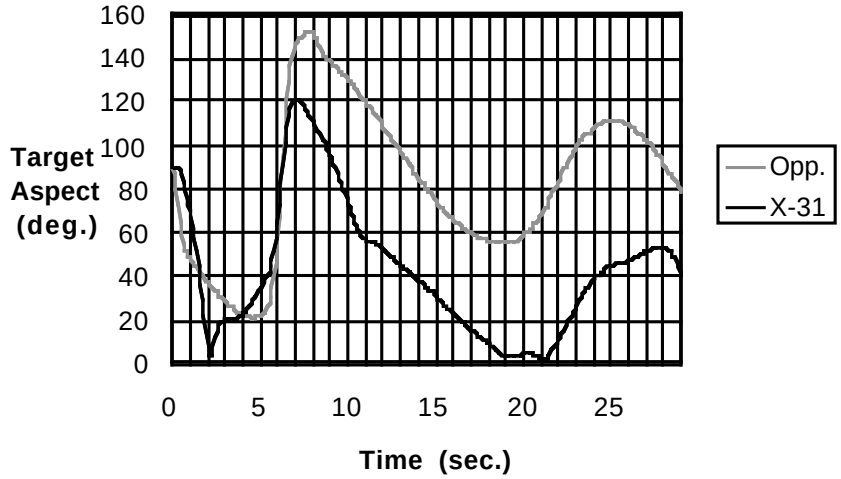


Figure G3.3.12. Baseline X-31 SSLA aspect angle time history.

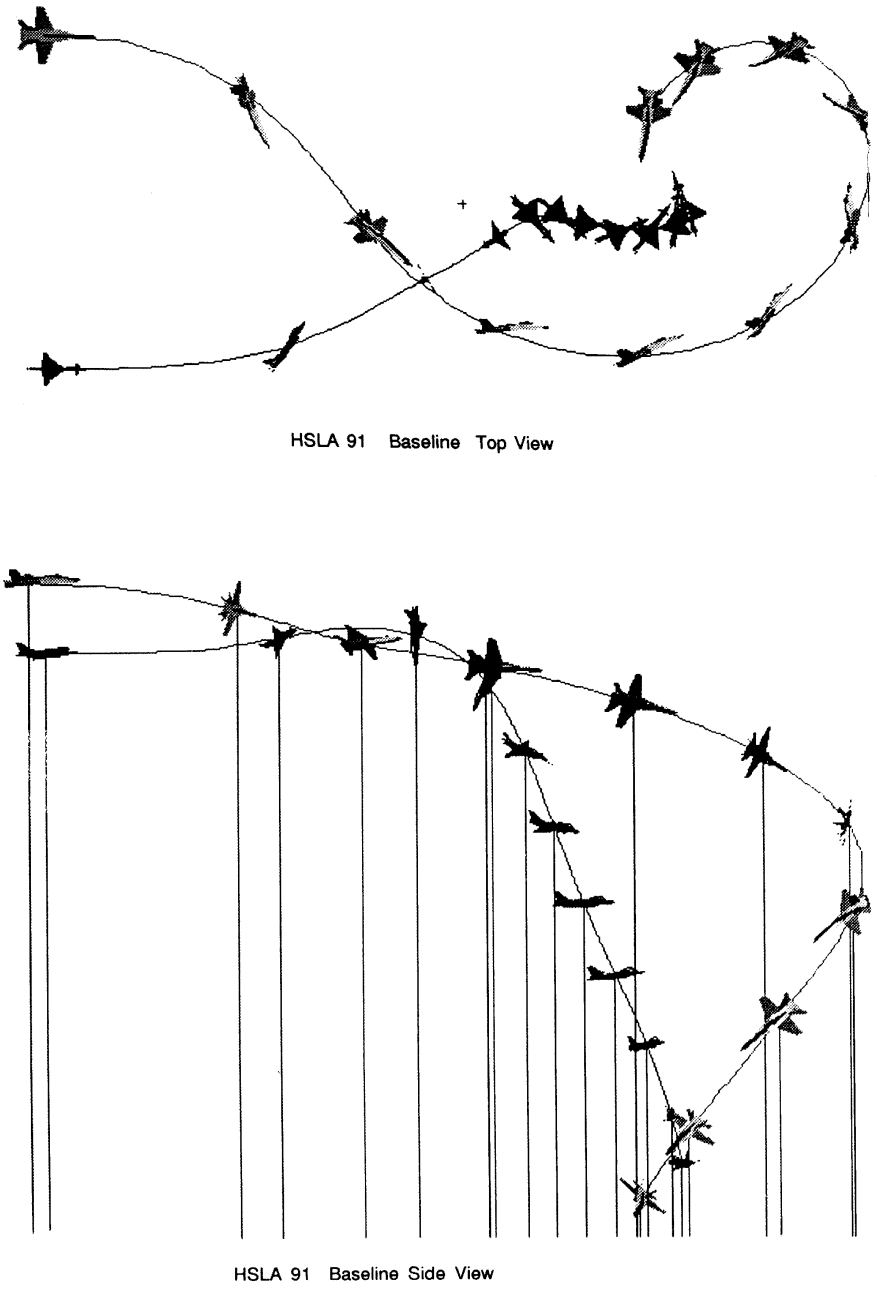
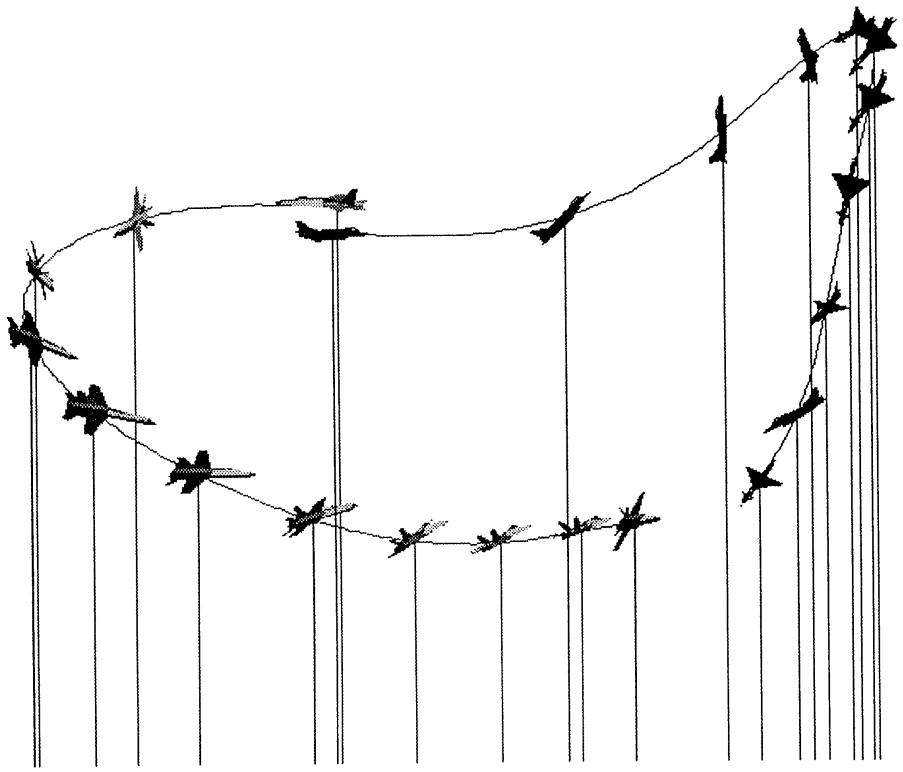


Figure G3.3.13. The baseline X-31 HSLA case plot.

1995). Several benefits were realized with the GLS approach. Unlike neural networks, GLS classifiers could be directly converted into descriptive instructions for pilots. The classifier syntax not only described the maneuver control sequence, but also specified the conditions, relative to the opponent, which trigger the controls. The GLS can also be run incrementally to accumulate rulebases from different sources, including human experts. Variations to aircraft performance, weapon system characteristics, and starting conditions are transparent, requiring no changes to the software or procedures. Opponent tactics can also be varied in AASPEM to test tactical sensitivities. The opponent aircraft tactics may even be developed by using the GLS in an alternating sequence with the X-31 aircraft.

A novel feature of the GLS is the ability to learn against any opponent. While this study focused on an AASPEM opponent, the GLS could also employ human opponents, using interactive or manned



B2BH 307 150% T/W 90 deg alpha Side View

B2BH 307 150% T/W 90 deg Alpha Top View

Figure G3.3.14. The X-31 with 150% T/W and 90° alpha B2BH case plot.

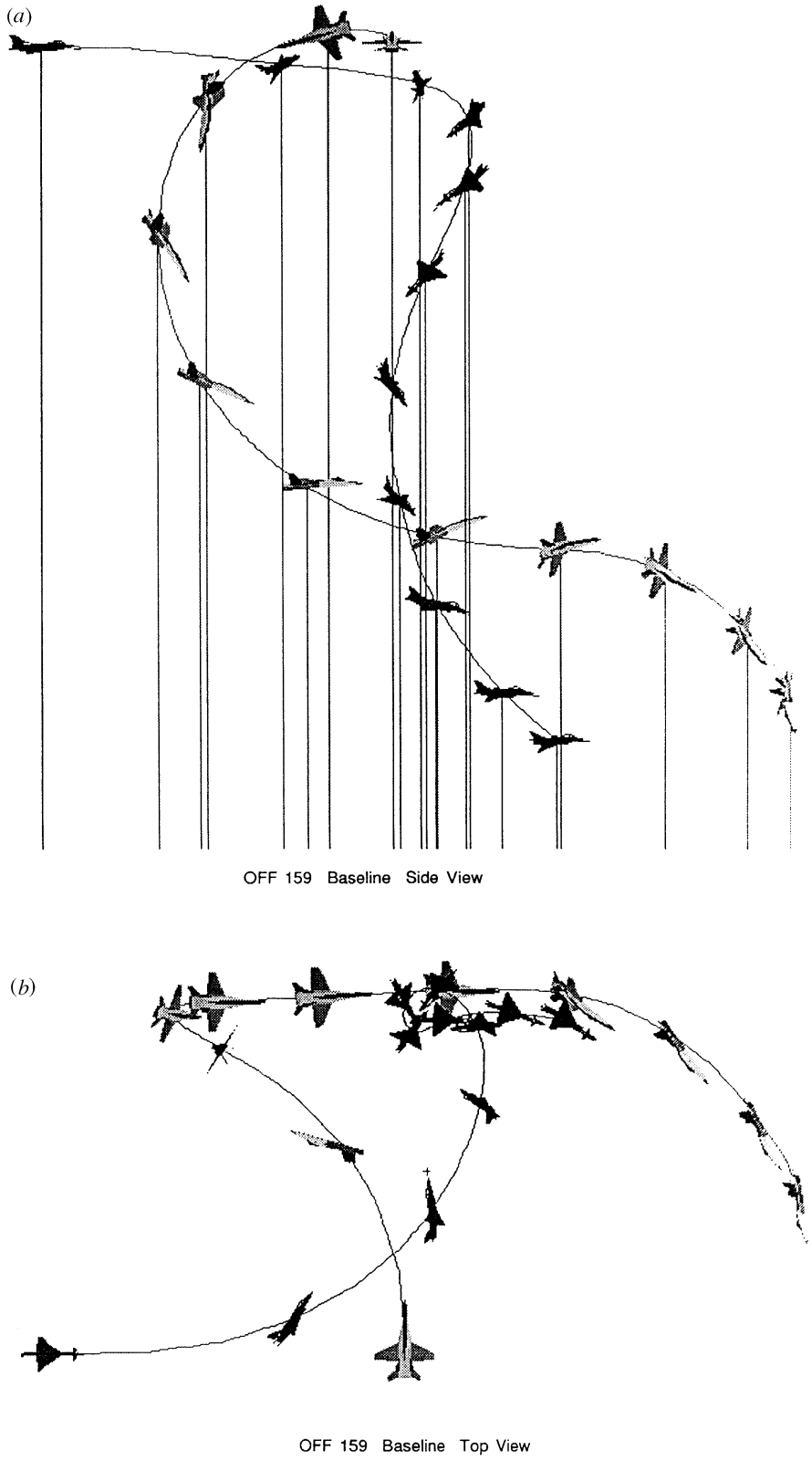
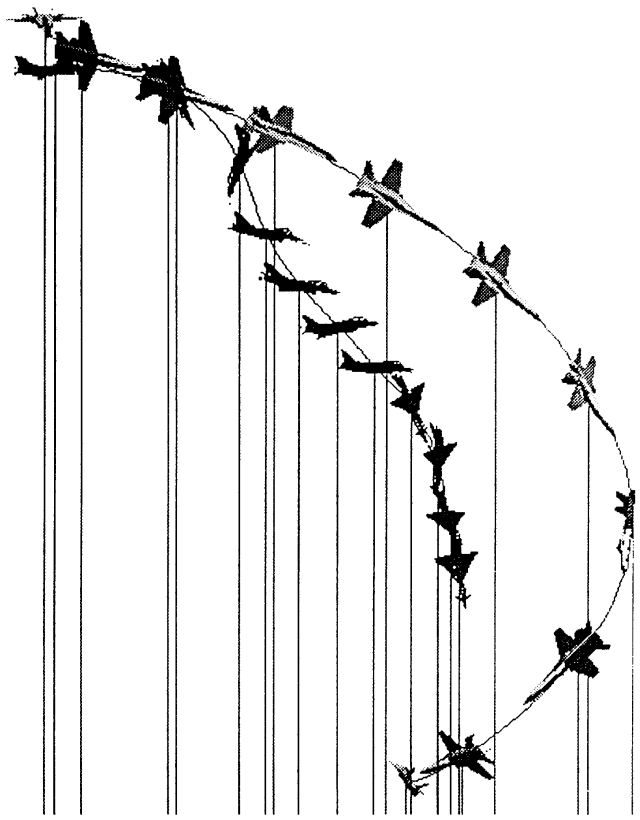


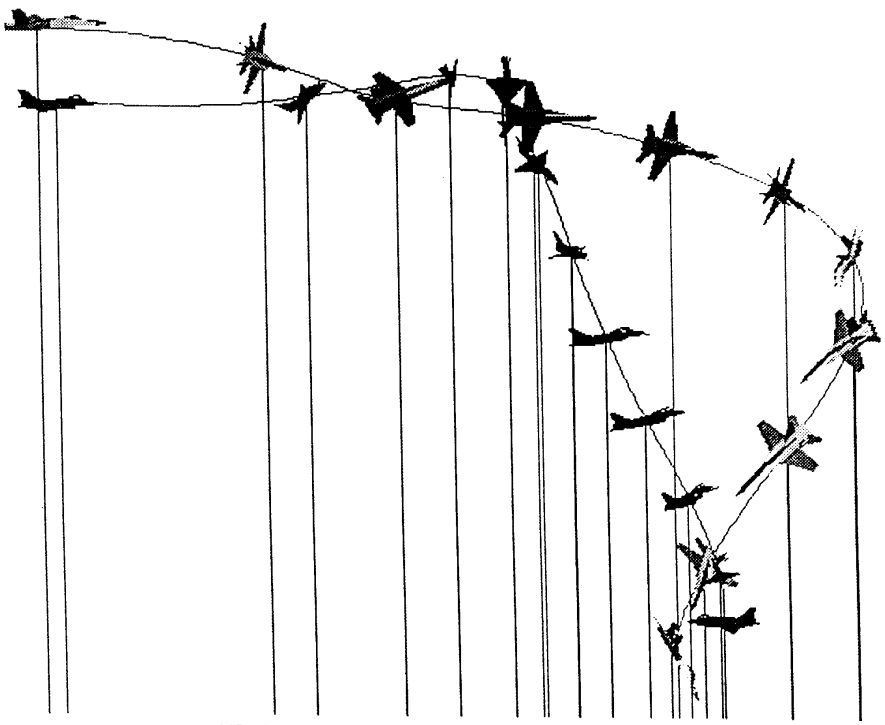
Figure G3.3.15. Plots of a variety of maneuvers discovered by the GLS.

(*c*)



DEF 59 Baseline Side View

(*d*)



HSLA 355 150% T/W 90 deg Alpha Side View

Figure G3.3.15. Continued.

simulation, for example. The GLS could be initially employed with AASPEM, and then face a manned opponent to encounter additional variability and higher skill levels.

These results represent a new application of machine learning to combat systems. Other potential uses of this capability include design evaluation, pilot training, tactical decision aids, and autonomous systems. GAs represent an efficient and effective tool for control and optimization of complex dynamic systems which would otherwise be intractable with conventional methods.

References

Doane P M, Gay C H, Fligg J A, Billman G, Situs K and Whiteford F 1989 *Multi-System Integrated Control (MuSIC) Program* Final Report, Wright Laboratories, Wright–Patterson AFB

Holland J H, Holyoak K J, Nisbett R E and Thagard P R 1986 *Induction: Processes of Inference, Learning, and Discovery* (Cambridge, MA: MIT Press)

Smith R E and Dike B A 1995 Learning novel fighter combat maneuver rules via genetic algorithms *Int. J. Expert Syst.* **8** 247–76

Further reading

1. Goldberg D E 1989 *Genetic Algorithms in Search, Optimization, and Machine Learning* (Reading, MA: Addison-Wesley)

2. Grefenstette J J 1989 A system for learning control strategies with genetic algorithms *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, June 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 183–90

3. Smith R E 1991 *Default Hierarchy Formation and Memory Exploitation in Learning Classifier Systems* TCGA Report 91003; PhD Dissertation, University of Alabama, University Microfilms 91-30 265

G3.4 Learning to break things: adaptive testing of intelligent controllers

Alan C Schultz, John Grefenstette and Kenneth De Jong

Abstract

Autonomous vehicles require sophisticated software controllers to maintain vehicle performance in the presence of vehicle faults. The test and evaluation of complex software controllers is a challenging task. The goal of this effort is to apply machine learning techniques to the general problem of evaluating an intelligent controller for an autonomous vehicle. The approach involves subjecting a controller to an adaptively chosen set of fault scenarios within a vehicle simulator and searching for combinations of faults that produce noteworthy performance by the vehicle controller. The search employs a genetic algorithm. The evidence suggests that this approach is an effective supplement to manual and other forms of automated testing of sophisticated software controllers. Several intelligent controllers were tested in this project using several different genetic-algorithm-based learning programs. Over the course of this research, the representation, evaluation function, genetic operators, and basic algorithm evolved as we tailored the algorithm to this class of problems. This case study presents this work from the point of view of describing the process that the authors followed in applying these learning algorithms to this real-world problem.

G3.4.1 Introduction

Autonomous vehicles require sophisticated software controllers to maintain vehicle performance in the presence of faults. The test and evaluation of such a software controller is a challenging task, given both the complexity of the software system and the richness of the test environment. The goal of this effort is to apply machine learning techniques to the general problem of evaluating a controller for an autonomous vehicle. The approach involves subjecting a controller to an adaptively chosen set of fault scenarios within a vehicle simulator and searching for combinations of faults that produce noteworthy performance by the vehicle controller. The search employs a genetic algorithm (GA), that is, an algorithm that simulates the dynamics of population genetics (Holland 1975, De Jong 1980, Grefenstette *et al* 1990, Schultz and Grefenstette 1992), to evolve sets of test cases for the vehicle controller.

We have illustrated the approach by evaluating the performance of two different intelligent controllers, one for an autonomous aircraft and the other for an autonomous underwater vehicle. The evidence suggests that this approach offers advantages over other forms of automated and manual testing of sophisticated software controllers, although this technique should supplement, not replace, other forms of software validation. This research is significant because it provides new techniques for the evaluation of complex software systems, and for the identification of classes of vehicle faults that are most likely to impact negatively on the performance of a proposed autonomous vehicle controller.

In this case study, we concentrate on describing the *process* of designing the learning approach as we apply it to this task, with particular emphasis on the evolution of the representation, the learning algorithm, and the evaluation function. In section G3.4.2, we will describe the task of testing intelligent controllers. In section G3.4.3, we briefly describe GAs and how they can be applied to this domain. Section G3.4.4

describes the first phase of this research, where a straightforward application of a generational GA is used on an aircraft controller. Section G3.4.5 describes the second phase of the project, where a GA-based rule learning system is applied to the testing of an autonomous underwater vehicle controller. The case study concludes with some general remarks about applying GAs to real-world problems.

G3.4.2 Testing the performance of an autonomous vehicle controller

Given a vehicle simulation and an intelligent, autonomous controller for that vehicle, what methods are available for testing the robustness of the controller? Validation and verification do not solve the problem of guaranteeing that the program will perform as desired. The controller may perform as specified, but the specifications may be incorrect; that is, the vehicle might not behave as expected. Testing all possible situations is obviously intractable due to the complexity of the system involved. Analysis techniques exist for testing the robustness of low-level controllers in isolation (see e.g. Appleby *et al* 1990), but the methods are not applicable to testing the vehicle as a whole.

Traditional approaches to performance testing of controllers can be labor intensive and time consuming. Some methods require that simulated vehicle missions be run with instantiated faults to test the robustness of the intelligent controller under various unanticipated conditions. To do this, the simulator would be altered to allow faults to be introduced into the vehicle simulation. Test engineers then hypothesize about the type of failures they anticipate to be a problem for the controller. After designing a *fault scenario* that will cause the particular failures to occur during a simulated mission, they observe the resulting behavior of the vehicle and then refine the fault scenario to better exercise the autonomous vehicle controller. This cycle is repeated until the test engineers are confident that the vehicle's behavior will be appropriate in the field.

Implicitly, the test engineers are performing a search of the space of fault scenarios looking for fault scenarios of interest. In this research, we developed a technique for automating the process of searching for interesting fault scenarios in the space of fault scenarios.

G3.4.3 Genetic algorithms

We wish to automate the process of creating and evaluating fault scenarios. To perform this search we will use a class of learning systems called *genetic algorithms* (GAs). In this system, a GA simulates the dynamics of population genetics by maintaining a knowledge base of fault scenarios that evolves over time in response to the observed performance in the vehicle simulation. The *fitness* of a fault scenario is captured by the evaluation function as described previously. The search proceeds by selecting fault scenarios from the current population based on fitness. That is, high-performing structures may be chosen several times for replication and poorly performing structures might not be chosen at all. Next, plausible new fault scenarios (*offspring*) are constructed by applying idealized *genetic search operators* to the selected structures. For example, *crossover* exchanges pieces of the representation of fault scenarios to create new offspring. *Mutation* makes small random changes to fault scenarios. The new fault scenarios are then evaluated in the next iteration (generation) of the algorithm.

B1.2
C3

G3.4.3.1 Applying the genetic algorithm

Figure G3.4.1 gives a diagrammatic view of how GAs can be applied to the problem of testing the performance of an intelligent controller. Given a vehicle simulator and an intelligent controller for the vehicle that is to be tested, the GA replaces the manual selection of new fault scenarios and automatically runs many scenarios, searching for interesting ones.

When applying GAs to particular problems, it is often necessary to tailor the algorithm to the chosen representation language, to develop new genetic operators that take advantage of available domain knowledge, and to develop an appropriate evaluation function. In the following sections, we describe the evolution of the algorithms used, and of the representation, evaluation function, and genetic operators. Figure G3.4.2 presents a process-oriented view of how the GA and the vehicle simulator effected changes in each other, and the source of domain knowledge that was applied during the process.

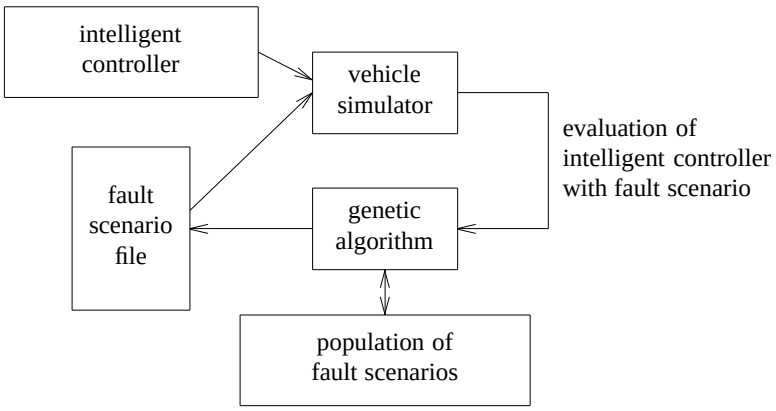


Figure G3.4.1. The GA replaces the human test engineer in deciding fault scenarios to apply to the intelligent controller under test.

G3.4.3.2 Evaluation of a fault scenario

When test engineers search the space of fault scenarios, they apply an evaluation criterion to provide a measurement of utility for fault scenarios in the search space. This evaluation criterion guides them in their search for scenarios of interest. To automate the search process, we needed to explicitly define an evaluation function that would provide the utility or fitness measure for each scenario examined. This is difficult, because evaluation criteria are often based on informal judgments. In this section, we describe the various approaches to defining evaluation functions that were considered in this project.

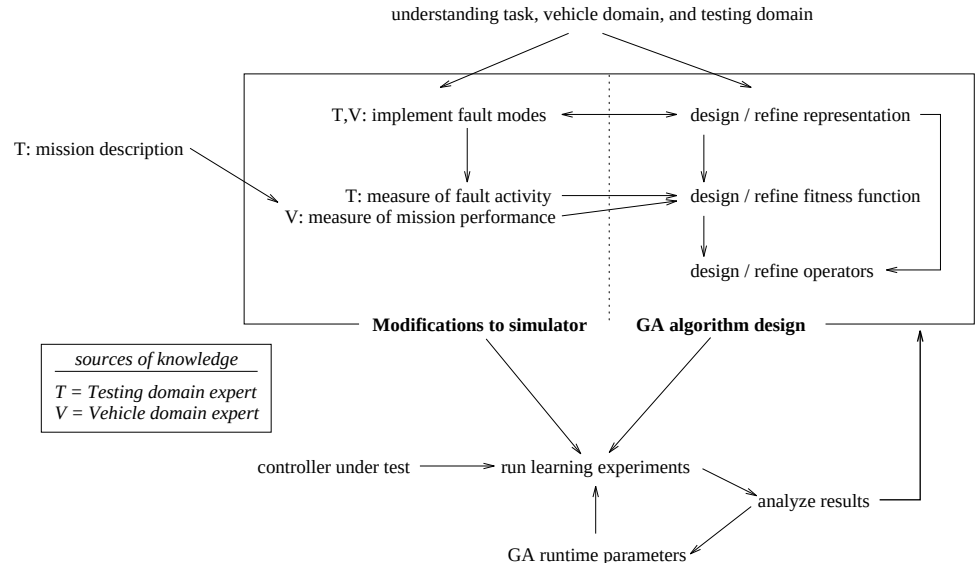


Figure G3.4.2. A process-oriented view of the application of a GA to this problem.

One approach is to define an evaluation function that would measure the difference between the actual performance of the autonomous controller on a given fault scenario against some form of ideal response. The ideal response could be approximated based on knowledge of the causal assumption behind the fault scenario (e.g. a certain sensor has failed, and should be recalibrated or ignored), or it could be based on the actions of an expert controller, or it could simply be to return to nominal performance of the mission plan in the least amount of time. The computation of the ideal response might rely on information that is not available to the controller under test. This approach has the advantage of yielding a more completely automated way of identifying problem areas for the autonomous controller, but it also requires a substantial effort to design software to compute an ideal response.

A second approach is to measure fitness on the basis of likelihood and the severity of the fault

conditions. The goal is to give the highest fitness to the most likely set of faults that cause the autonomous controller to degrade to a specified level. This approach is useful when probability estimates of the various fault modes are available to be used in constructing the evaluation function. Unfortunately, accurate probability estimates are difficult to obtain, especially for vehicles with few or no prior real-world data.

A third approach is to define an evaluation function that rewards fault scenarios that occur on the boundary of the performance space of the autonomous controller. That is, a set of fault rules would receive high fitness rating if it causes the controller to degrade sufficiently, but some minor variation does not. Such a fitness function would facilitate the identification of 'hot spots' in the performance space of the autonomous controller. The computation of such a fitness function would require the evaluation of several scenarios for each fault specification, and, given the computation cost involved in each evaluation of the controller, which requires a complete simulation of a mission, this approach was not feasible.

A fourth approach to constructing effective evaluation functions is to define and search for scenarios that are *interesting*. There are several possible ways to define interesting in the context of an intelligent controller, each giving a separate evaluation function. One interesting class of scenarios is those in which minimal fault activity causes a mission failure or vehicle loss. The dual of this class is the class of scenarios in which maximal fault activity still permits a high degree of mission success.

Using this fourth approach, we have implemented evaluation functions for the two controllers used in this study. This approach proved helpful in qualitatively examining the overall performance profile on an autonomous vehicle controller.

G3.4.4 Phase 1: testing an aircraft controller

Our initial development focused on experiments with a controller for an autonomous air vehicle. In these experiments, the controller was tested using a medium-fidelity, three-dimensional simulation of a jet aircraft. The task for the controller is to fly to and land on an aircraft carrier. The simulation includes the ability to control environmental conditions, in particular, constant wind and wind gusts.

G3.4.4.1 Description of the vehicle controller

The autonomous controller, which is responsible for flying the aircraft and performing the landing on the carrier deck, was designed using a subsumption architecture approach (Hartley and Pipitone 1991). The controller is composed of individual behaviors, operating at different levels of abstraction, that communicate among themselves and together allow the aircraft to fly and to land. Top-level behaviors include *fly-craft* and *land-craft*. At a lower level, behaviors include *fly-heading* and *fly-altitude*. The lowest-level behaviors include *hold-pitch* and *adjust-roll*. After the initial design, optimization techniques were used to improve the controller such that it was successful in flying and landing the aircraft, even in conditions of constant wind and wind gusts.

To incorporate a learning algorithm, the simulator was modified to allow various faults to be instantiated in the aircraft. In addition, the simulator was modified to read a file at startup that contains a fault scenario. The simulator first reads the initial conditions and configures the starting state, and then reads the fault rules for use during the simulation. Each cycle of the simulation, the rules are tested to see whether a fault should be instantiated into the system.

G3.4.4.2 Initial representation of fault scenarios

In our approach, a fault scenario is a description of faults that can occur in a vehicle, and the conditions under which they will occur. Furthermore, the fault scenario might include information about the environment under which the vehicle is operating. This section describes our initial representation of a fault scenario in detail.

Figure G3.4.3 shows a representation of a fault scenario. A fault scenario is composed of two main parts, the *initial conditions* and the *fault rules*. The initial conditions give starting conditions for the vehicle and environment in the simulator, such as vehicle altitude, initial speed, attitude and position. The initial conditions are read when the simulator starts up and the associated elements of the environment or vehicle are set accordingly. The fault rules are the rules that map current conditions (i.e. the state of the vehicle and environment) to fault modes to be instantiated.

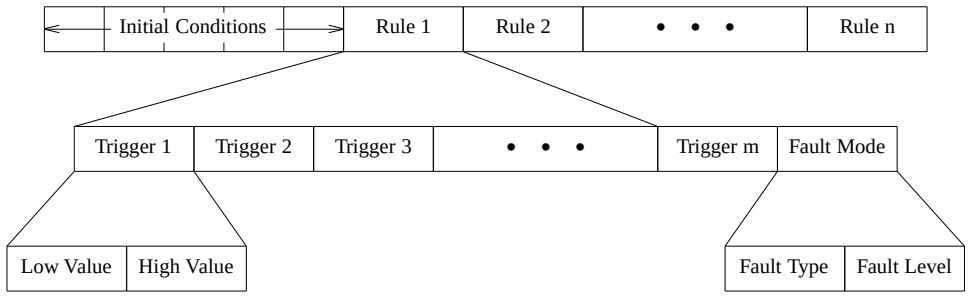


Figure G3.4.3. A representation of a fault scenario.

Each rule is composed of two parts, *triggers* and the *fault mode*. The triggers make up the rule antecedent, and represent the conditions that must be met for the fault to occur. When the conditions specified by the triggers are met, the fault mode is instantiated in the vehicle simulation.

Each of the triggers measures some aspect of the current state of the vehicle, the environment, or the state of other faults that might be activated at that time. Each trigger is composed of a low value and a high value, and if the measured quantity in the state is within the range of the trigger, then that trigger is said to be satisfied. All triggers in a rule must be satisfied for the fault to be triggered.

A fault mode, or right-hand side of a fault rule, has two parts, a *fault type* and a *fault level*. The fault type describes the subsystem that will fail in the vehicle model. The fault level is a parameter that describes the severity of the failure.

To summarize, a fault scenario is used as follows. At the start of a simulated mission, the initial conditions are first read, and those variables are set in the simulation. At each time step in the simulation, each rule is examined to see if the triggers are satisfied, and if they are, then that rule's fault mode is instantiated with the given amount of degradation.

G3.4.4.3 Modeling faults in the vehicle

Three classes of faults were introduced into the vehicle simulation, *control* faults, *sensor* faults, and *model* faults. Control faults occur when a controller commands an action by an actuator but the actuator fails to perform the commanded action. Control faults have been modeled for the elevators, rudders, ailerons, and flaps. Sensor faults represent failures of sensors or detectors of the vehicle. In these cases, the controller tries to read a sensor, and receives erroneous information because of either noise or sensor failure. Sensor faults have been modeled for the vehicles sensors for measuring pitch, yaw, and roll. Model faults are failures of the vehicle that are not directly related to sensors or effectors, and usually involve physical aspects of the vehicle. For example, in an autonomous underwater vehicle instantiating a leak is a model fault. There is one model fault in this vehicle simulation, drag, which represents a change in the parasitic drag of the vehicle, as if a structure of the vehicle were damaged, resulting in increased drag.

In addition to these three classes of faults, faults can also be identified as *persistent* or *nonpersistent*. Persistent faults, once instantiated, do not cease, while nonpersistent faults must be reinstantiated at each time step to continue. For example, actuators and sensors tend to have intermittent failures, and can return to a fault-free state, and therefore would be modeled as nonpersistent. On the other hand, increased drag due to damage of the vehicle's body cannot be undone and is modeled as a persistent fault. Details of the modeling of the faults can be found in the article by Schultz *et al* (1993).

G3.4.4.4 Trigger conditions for the faults

In the initial experiments, there are 21 triggers (conditions) for each fault rule. Some of the triggers measure the state of the aircraft and others examine other fault conditions. The triggers include the three components of the velocity vector, the three values for the vehicle's absolute position in space, the attitude of the vehicle (pitch, yaw, and roll), the current flap setting, the current thrust setting, the elapsed time since the mission began, the time since the last fault was instantiated, and the current state of each of the faults: currently active, currently not active, or not important (i.e. does not matter).

G3.4.4.5 Setting initial conditions

The first group of items in the fault scenario file is the initial conditions, which configure the starting state of the simulator. The range of initial conditions was restricted so that no setting of these conditions can by itself cause the vehicle to fail. When the simulation starts, the aircraft begins its mission approximately two nautical miles from the carrier and then proceeds to land. The initial conditions control environmental conditions and the exact starting configuration of the aircraft, including wind speed, wind direction, aircraft altitude, distance of the aircraft from the carrier, how well the aircraft is lined up with the carrier initially, and the initial forward velocity of the aircraft. The following code shows part of a fault scenario file for this system.

```
**** Initial Conditions **********************
  set        wind speed = 8
  set    wind direction = 58
  set          altitude = 1460
  set          velocity = 121
**** Rule 1 **********************************
IF   -63.00  <=  velocity[x] <=   64.00   AND
     -56.00  <=  velocity[y] <=  -28.00   AND
    -254.00  <=  velocity[z] <= -224.00   AND
     -16220  <=  position[x] <=    -860   AND
         13  <=  position[y] <=     832   AND
       2040  <=  position[z] <=    2040   AND
       -325  <=        pitch <=     635   AND
      -1370  <=          yaw <=    -100   AND
.
.
.
THEN set fault type  = S_roll
     set fault value = -0.232

**** Rule 2 **********************************
IF  . . .
```

G3.4.4.6 Evaluation function

The role of the evaluation, or fitness, function in a genetic algorithm is to provide a measurement of utility for arbitrary points in the search space defined by the representation language. For these experiments, we adapted the fourth approach in section G3.4.3.2: define and search for scenarios that are *interesting*.

For the testing of the aircraft controller, we have defined an evaluation function that gives high ratings to scenarios that induce interesting behaviors by the vehicle controller. Maximizing the evaluation function searches for failures of the aircraft controller in the face of minimal vehicle failures. This searches for interesting weaknesses of the aircraft controller. Minimizing this function searches for successes of the aircraft controller in light of significant vehicle failures. This allows us to characterize the robustness of the controller with respect to some general classes of faults.

We begin by defining *fault activity*. First, the absolute values of the fault levels active during a given time step are normalized so that they are between one and ten and then the product is taken:

$$\text{current fault activity} = \prod_{\text{active rules}} ((|\text{fault level}| \times 9.0) + 1.0).$$

Then we take the average fault activity over the entire mission:

$$\text{fault activity} = \frac{\sum_{\text{time}} \text{current fault activity}}{\text{time}}.$$

The fault activity measures the level of faults that are introduced over the entire length of a mission.

The simulator also returns a score based on the quality of the landing using factors such as the distance from the center line, which cable the aircraft's tail hook caught, the roll angle at touchdown, and velocity

of descent, and returns a score as follows:

$$\text{score} = \begin{cases} 1 & \text{if CRASH LANDING} \\ 2 & \text{if ABORT} \\ 3 \rightarrow 10 & \text{if SAFE LANDING.} \end{cases}$$

Therefore score ranges between 1, which indicates a crash, and 10, which indicates a perfect landing. We now combine the fault activity and the score as follows:

$$\text{eval} = \frac{1}{\text{fault activity} \times \text{score}}.$$

When no faults occur yet a crash landing is experienced (actually, this is impossible by design), then *eval* returns 1, the maximum value possible. With maximal fault levels throughout the mission and a perfect landing, *eval* returns 0.01, the minimal value possible.

To find the first class of interesting scenarios, those where minimal fault activity results in failure of the intelligent controller, we use the GA to maximize *eval*. To find the second class of scenarios, those where, despite maximal fault activity, the aircraft still manages to land well, we use the GA to minimize *eval*.

G3.4.4.7 Results of experiments in phase one

The results of the initial experiments can be found in the article by Schultz *et al* (1993), and are summarized here. In all experiments, we used a population size of 100, and ran the GA for 100 generations resulting in 10 000 total evaluations. We first maximized the evaluation function to find several minimum-fault, maximum-failure scenarios on the simulator. The GA quickly homed in on scenarios with high fitness, that is scenarios where minimal fault activity led to controller failure.

By examining the scenarios identified as interesting by the GA, we were able to draw the following general conclusions about the intelligent controller:

- Roll control was most critical at the start of the touchdown phase.
- Sensor errors were much harder to recover from than control errors.
- Even slight increases of drag caused the controller to behave poorly.

Next, we minimized the evaluation function to search for successes of the intelligent controller in light of significant vehicle failures. In this case, we have characterized the robustness of the controller with respect to some general classes of faults:

- The GA again found that the controller could recover from control faults, but that sensor faults were much harder to handle.
- Recovery from faults that affected the pitch of the aircraft were easier than recovery from faults affecting the roll of the craft. This agrees with the earlier observation.
- Finally, the GA identified situations in which it was possible for some faults to 'cancel' the effects of other faults (e.g. positive sensor errors may offset negative control errors).

In particular, the scenarios as a group tend to indicate classes of weaknesses, as opposed to only highlighting single weaknesses. This allows the controller designers to improve the robustness of the controller over a class as opposed to only patching specific instances of problems.

G3.4.5 Phase 2: testing an autonomous underwater vehicle

The second phase of the project involved scaling up the methodology to start testing vehicle controllers for the Charles S Draper Laboratories (CSDL) autonomous underwater vehicle (AUV). CSDL supplied a simulation of the vehicle that was to be used for the testing effort, which had already been instrumented with hooks for instantiating faults into the vehicle.

G3.4.5.1 Use of SAMUEL learning system

To support the anticipated scaleup to more complex fault scenarios, and for better representation of temporal aspects of the fault scenarios, we started using the SAMUEL rule learning system (Grefenstette *et al* 1990). SAMUEL differs significantly from the earlier algorithm used in phase 1 of the project. In particular, SAMUEL was designed to learn sequential decision rules for solving sequential decision problems. The rule representation was more appropriate for this domain. Also, the algorithm was designed to consider temporal aspects of the problem.

G3.4.5.2 Changes in the representation

Conceptually, the representation is similar to phase 1, except, here, rules sets are the internal representation used by SAMUEL. Also, attached to each rule is a rule strength (not shown here) that indicates the utility of the rule compared to others in similar situations (i.e. how well this rule does in solving the problem compared to other rules). When more than one rule matches during a decision cycle, the rule strength is used to determine the rule that will actually instantiate a fault. In addition, a credit assignment algorithm is used to update the strength of the rules based on the outcome of a simulated mission. Some rules from a SAMUEL rule set for this domain are:

```
RULE 4
IF   depthrate = 25
     AND  heading = [0, 355]
     AND  v_ballast3 > 2
THEN SET fault = any (severity = [0, 1])

RULE 8
IF   temp_mc_fhs < 100
     AND  temp_v_sensr < 86
THEN SET fault = hull_flt8 (severity = [0, 1])
```

To be more expressive, and more suited to this task, several changes were made to SAMUEL's rule language. The rule language used by SAMUEL was expanded to allow more natural specifications of conditions and actions. Relational operators (see rule 8 above) were added. Rules may also include real-valued conditions and actions, relaxing the previous restriction to integer-valued attributes. The actions in SAMUEL rules may now include *qualifiers* that serve as parameters for the action values. Previously, the fault severity was treated as a separate action. This difference is important to SAMUEL; the new form of the rule links the fault mode and its severity.

G3.4.5.3 Addition of Lamarckian learning operators

One property of the CSDL simulator and the test missions we used was that each simulation run took up to 10 minutes. To use our technique, we needed to maximize the amount of information learned in each trial. One approach to accomplish this involved using *Lamarckian operators* in addition to the genetic operators. Operators, such as *generalization* and *specialization*, are triggered when appropriate situations occur during a run. For more information on Lamarckian operators, see the article by Grefenstette (1991).

G3.4.5.4 Changes in the fitness function

A measure of the vehicle controller's performance is calculated by CSDL based on the position of the vehicle with respect to the commands given to the controller (e.g. deviation from the commanded waypoint). This value, vc score, has the range $0 \rightarrow 100$. We remap that value to $1 \rightarrow 10$, and also emphasize the differences on the high end of the range by cubing the raw vc score. This allows SAMUEL to exploit small differences in the performance of the controller. The result is the *vehicle performance*:

$$\text{vehicle performance} = \left((\text{vc score}/100)^3 \times 9\right) + 1.$$

To measure the level of fault activity, we begin by realizing that some of the fault modes are modeled as persisting over time. In some cases, a fault severity can increase, but cannot decrease. Other fault modes are binary; any fault severity level higher than zero will result in failure.

To handle these cases, we track the current fault level for all persistent faults, and binary faults are always assigned a severity level of one if the value returned by the fault scenario is greater than zero. We then sum up all faults that are active, over the entire episode, and normalize by the total number of faults we could have experienced. We emphasize the lower end of the scale by taking the square root, and then map this to the range of $1 \rightarrow 10$. In general, few faults are actually active compared to the total amount of fault activity that could be present, and this emphasis of the smaller values allows SAMUEL to exploit small difference in the fault activity level. The result is the *fault activity*:

$$\text{fault act} = \left(\left(\frac{\text{fault sum}}{(\text{perm faults} + 1)(\text{steps} + 1)} \right)^{1/2} \times 9 \right) + 1.$$

We now produce the final evaluation function as follows:

$$\text{fitness} = [1/(\text{vehicle performance} \times \text{fault activity})]^{1/2}.$$

This is similar to the fitness function used in the Genesis experiments. Maximizing this function will result in minimal fault activity causing bad vehicle performance. Minimizing the function will find scenarios that allow successful missions despite a high level of fault activity. Results obtained in recent experiments indicate that this evaluation function works well in finding interesting fault scenarios.

G3.4.6 Conclusion

In applying GAs to any real-world problem, careful consideration must be given to the choice of representation, the modification of operators, and the design of the evaluation function.

In many problems, there are alternative methods for representing the problem space. However, some representations are better suited for search by GAs than other representations. A classic example of a representational error is using the space of combinations to represent problems that are better suited for representation in the space of permutations (e.g. the traveling salesperson problem). Problems in the representation can be rather subtle. Using an incorrect coding might result in 'Hamming cliffs' in mapping between the genetic encoding and the search space. The result of a poorly chosen representation is that the genetic search operators may have difficulty in transforming one candidate solution to another solution that would be 'nearby' in the search space. In the work discussed here, a high-level representation was chosen to be close to the concepts used by the actual domain experts. Using high-level representations in this domain ensure that the search operators can effectively search the space of candidate solutions.

The important point learned is that the GA practitioner and the domain expert must work closely in designing a representation that captures the correct aspects of the domain, correct in that the genetic operators can properly search that space. Generalized to machine learning, representation has always been an important aspect of solving a hard problem; the algorithm and representation are closely tied to one another, and a poor choice of representation can reduce (or even eliminate) positive results.

Once the representation is chosen, the genetic operators (e.g. crossover and mutation) may have to be modified, or new operators added. For example, if the space of permutations is being searched, special crossover operators might be used to guarantee that offspring are still legal representations. In this domain, special operators were introduced to match the high-level representation, and to speed up the learning.

In general, when applying GAs to real-world problems, it is usually beneficial to add to the basic operators, using domain-specific knowledge to introduce new operators. In machine learning, weak methods can generally benefit from the addition of domain knowledge, if it is practical to add that knowledge to the system.

Finally, GAs require an evaluation function that is reasonable to compute. In this project, the time per evaluation represented a serious problem. A complete simulation run of the vehicle with the controller was required for each trial. Given that 10 000 trials might be necessary and that a single trial could take 10 minutes, the total computation time could take several months. This issue was addressed in several ways. First, special Lamarckian operators were introduced to speed up learning. Operators that perform generalization and specialization of rules based on traces of recent episodes were added to the usual genetic operators. Second, parallelism was introduced by using a distributed network of computers. Third, we have started to examine ways to extract several learning episodes from a single mission of the vehicle.

The actual application of GAs to real problems takes careful coordination with the domain experts to successfully solve the problem. Defining and refining the representation, operators, and evaluation function is itself an evolutionary process that by necessity must include the domain experts.

References

Appleby B, Bonnice W and Bedrossian N 1990 Robustness analysis methods for underwater vehicle control systems *Symp. on Underwater Vehicle Technology (Washington, DC)* (Piscataway, NJ: IEEE) pp 74–80

De Jong K A 1980 Adaptive system design: a genetic approach *IEEE Trans. Syst. Man Cybernet.* **SMC-10** 566–74

Grefenstette J J 1991 Lamarckian learning in multi-agent environments *4th Int. Conf. on Genetic Algorithms (San Diego, CA, July 1991)* ed R K Belew and L B Booker (San Mateo, CA: Morgan Kaufmann) pp 303–10

Grefenstette J J, Ramsey C L and Schultz A C 1990 Learning sequential decision rules using simulation models and competition *Machine Learning* **5** 355–81

Hartley R L and Pipitone F J 1991 Experiments with the subsumption architecture *IEEE Int. Conf. on Robotics and Automation (Sacramento, CA)* (Los alamitos, CA: IEEE Computer Society Press) pp 1652–9

Holland J H 1975 *Adaptation in Natural and Artificial Systems* (Ann Arbor, MI: University of Michigan Press)

Schultz A C and Grefenstette J J 1992 Using a genetic algorithm to learn behaviors for autonomous vehicles *AIAA Guidance, Navigation and Control Conf.* (Washington, DC: AIAA) pp 739–49

Schultz A C, Grefenstette J J and De Jong K A 1993 Test and Evaluation by Genetic Algorithms *IEEE Expert* **8** 9–14

G3.5 Population-oriented simulated annealing: an evolutionary–thermodynamic hybrid approach to VLSI network partitioning

J M Varanelli, James P Cohoon and W N Martin

Abstract

Very large-scale integration (VLSI) network partitioning is an important step in the automatic layout of VLSI digital circuits. Since the VLSI network partitioning task is NP-hard, a number of heuristic approaches have been described in the literature. Two of the most effective of these approaches are the genetic algorithm and simulated annealing. However, each of the methods has drawbacks. In an attempt to minimize these drawbacks while maintaining each paradigm's particular advantages, an evolutionary–thermodynamic hybrid algorithm called POSA (population-oriented simulated annealing) is introduced. This algorithm provides the dynamic convergence control lacking in traditional evolutionary approaches such as the genetic algorithm while allowing for efficient parallelization, which is difficult for most simulated annealing applications. Experimental results are presented that indicate the heuristic compares favorably to existing VLSI network partitioning methods. Results are also presented for a prototype distributed version of the algorithm.

G3.5.1 Introduction

As the transistor densities of digital integrated circuits have steadily increased to today's levels—of the order of tens of millions of transistors on a single chip—the automation of the VLSI design process has become of paramount importance in order to maximize designer productivity and minimize development time. One of the final steps in this design process is called *physical design layout*. Layout is the actual arrangement and electrical interconnection of the circuit elements, or cells, of the given logical design onto the chip surface. A group of *cells* specified to be electrically interconnected is called a *network*, or *net*. The task of arranging the cells on the chip surface is typically referred to as *placement*, while the physical interconnection of the networks is called *routing*.

An early observation in the development of placement algorithms is the fact that cells contained on the same network should normally be placed close together to minimize the routing area between them. This observation led to the concept of *VLSI min-cut partitioning* (Dunlop and Kernighan 1985). The min-cut approach involves the repeated bipartitioning of the set of networks described by the logical design. At each step in the process, the goal is to minimize the number of specified networks that have cells in both subsets described by the partition; or in other words, to minimize the number of networks that are *cut* by the partition. It is this problem of bipartitioning a specified set of circuit networks, or the *VLSI network partitioning problem* (VLSI-NPP), that serves as the focus of this discussion.

The VLSI-NPP has been shown to be NP-hard (Garey and Johnson 1979). It is a generalization of the *graph partitioning problem* (GPP) (Fiduccia and Mattheyses 1982, Kirkpatrick *et al* 1983). A number of effective heuristic VLSI-NPP approaches have been proposed (Fiduccia and Mattheyses 1982, Davis 1987, Kernighan and Lin 1970, Kirkpatrick *et al* 1983). This discussion introduces a new heuristic approach to the VLSI-NPP called POSA, short for *population-oriented simulated annealing* (Goldberg 1990, Varanelli

and Cohoon 1995, Varanelli 1996). It is an evolutionary–thermodynamic hybrid technique designed to incorporate the advantages of evolutionary computation paradigms, i.e. the *genetic algorithm* (GA), with the advantages of thermodynamic paradigms, i.e. *simulated annealing* (SA). Both methods have proven to be quite successful at solving VLSI-NPPs (Aarts and Korst 1989, Fiduccia and Mattheyses 1982, Davis 1987, Kirkpatrick *et al* 1983, van Laarhoven and Aarts 1987, Varanelli and Cohoon 1995, Varanelli 1996).

The remainder of this discussion is organized as follows. Section G3.5.2 formally introduces the VLSI-NPP. Section G3.5.3 describes the motivation behind early evolutionary–thermodynamic hybrids and introduces the POSA heuristic. Section G3.5.4 presents experimental results for two benchmark and two random VLSI circuits, including a solution quality analysis based on a sequential version of the algorithm and a speedup and scalability analysis based on a distributed version executed over a network of ten workstations.

G3.5.2 The VLSI network partitioning problem

The input to the VLSI-NPP is the logical description of the circuit being considered. Its description consists of a list of cells and a specification of the electrical networks. Each cell has an associated positive-valued area and it is assumed that each network represents the interconnection of at least two cells.

The goal of the VLSI-NPP is to partition the specified set of cells $\mathcal{C}$ into two disjoint subsets $\mathcal{A}$ and $\mathcal{B}$ such that the number of nets with cells in both subsets is minimized. The partitioning is constrained by the requirement that the sums of the areas of the cells in each subset must be kept equal. The difference in total cell area between the two subsets is used as a penalty term, resulting in the following objective function to be minimized:

$$f(\mathcal{P}) = |E_{\text{cut}}(\mathcal{P}, \mathcal{N})| + \omega \cdot \left(\sum_{a \in \mathcal{A}_\mathcal{P}} \text{area}(a) - \sum_{b \in \mathcal{B}_\mathcal{P}} \text{area}(b) \right)^2 \qquad \text{(G3.5.1)}$$

where $\mathcal{P} = \{\mathcal{A}_\mathcal{P}, \mathcal{B}_\mathcal{P}\}$ is a partition of $\mathcal{C}$, $\mathcal{N}$ is the given set of nets, $|E_{\text{cut}}(\mathcal{P}, \mathcal{N})|$ is the number of nets with cells in both $\mathcal{A}_\mathcal{P}$ and $\mathcal{B}_\mathcal{P}$ and ω is a weighting constant. An appropriate value for ω depends upon certain other problem-related design choices made during algorithmic development that will be described here.

The most important of these decisions is the choice of a suitable variation mechanism for creating new candidate solutions from the current solution. There have typically been two generation mechanisms employed by previous VLSI-NPP heuristics. The first method is a cell interchange introduced by Kernighan and Lin (1970) in which one cell from each subset is chosen according to some criterion and their locations are swapped. The other common VLSI-NPP generation mechanism is the movement of a single cell from one subset to another, first introduced by Fiduccia and Mattheyses (1982). Although no particular advantages for either method have been demonstrated in the literature, the single-cell method was chosen for the POSA heuristic presented in the next section due to its smaller neighborhood size. An example of this type of variation mechanism is shown in figure G3.5.1.

Another important design decision is whether to consider infeasible solutions in the search process. An *infeasible* solution is one that does not meet a predetermined *balance criterion*. Typically there is a *balance tolerance* associated with feasible-only approaches (Fiduccia and Mattheyees 1982). Only feasible partitions are considered for the POSA heuristic. Candidate solutions must be generated until one is found that is balanced to within the stated tolerance bounds. This additional overhead in solution generation is counterbalanced by a constricted search space. Common balance tolerances range from the size of the largest cell in the circuit to as much as 5% of the total cell area. The balance tolerance for the POSA heuristic is the size of the largest cell.

As mentioned above, the choice of the weighting constant ω in equation (G3.5.1) depends upon the design choices just described. For the purposes of this discussion, $\omega = 1/(4{C_{\max}}^2)$, where $C_{\max}$ is the area of the largest cell in the circuit. This value of ω is chosen such that the penalty term will always be less than one, guaranteeing priority to solutions with fewer cut networks. This choice for ω is consistent with other VLSI-NPP investigations.

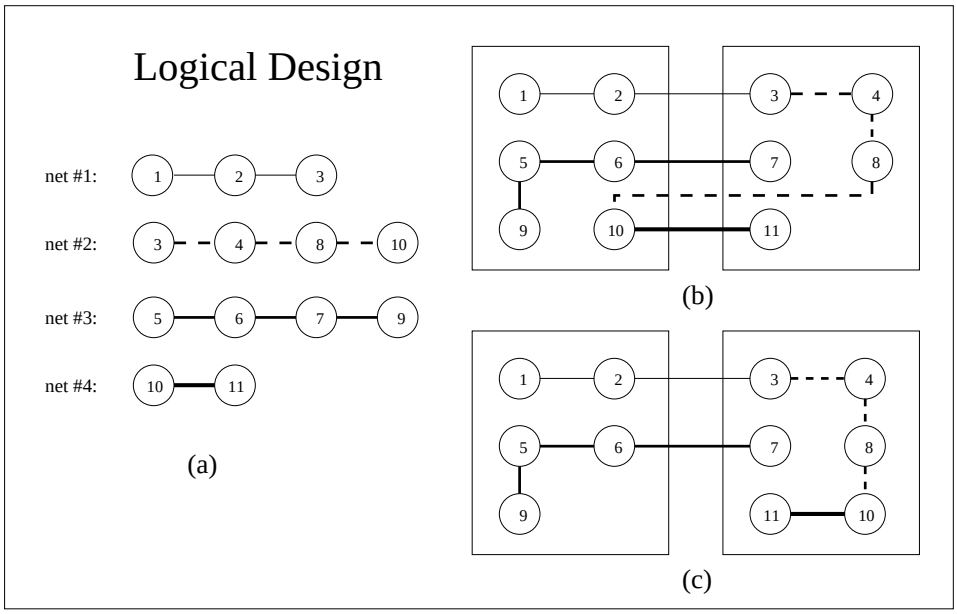


Figure G3.5.1. The VLSI-NPP using a single-cell variation mechanism. (a) This example has 11 cells (circles) connected by four networks (lines). (b) A solution with all four networks cut by the partition. (c) When cell 10 is moved, two networks are no longer cut.

G3.5.3 The POSA approach

This section introduces the POSA heuristic for the VLSI-NPP. First, motivations for developing evolutionary–thermodynamic hybrid algorithms and some previous hybrid systems are described. Then the POSA algorithm for the VLSI-NPP is given in detail. Finally, various implementation details are discussed for both the sequential and distributed POSA systems used to generate the experimental results presented in section G3.5.4.

G3.5.3.1 Genetic algorithm–simulated annealing hybrid systems

Evolutionary computation paradigms such as the GA have received considerable attention as general-purpose stochastic optimization techniques (Davis 1987, Holland 1975). In particular, the GA has proven to be quite successful at solving VLSI-NPPs (Davis 1987, Varanelli and Cohoon 1995, Varanelli 1996). However, this success comes at a typically high computational cost compared to other VLSI-NPP methods. The high computational cost is associated with the difficulty in controlling GA *convergence*. The convergence control problem seen in GAs is alleviated somewhat by the fact that the population-oriented approach of the GA allows for explicit parallelization of the process (Davis 1987, Goldberg 1990, Holland 1975, Mahfoud and Goldberg 1995). B2.2.5

SA is another general-purpose stochastic optimization technique that has proven to be quite effective for solving VLSI-NPPs (Aarts and Korst 1989, Azencott 1992, Davis 1987, Kirkpatrick *et al* 1983, van Laarhoven and Aarts 1987, Varanelli and Cohoon 1995, Varanelli 1996). The main advantage of SA over the GA is the convergence control afforded by the chosen temperature schedule. It allows the user to determine the appropriate tradeoff between computation time and final solution quality. Like the GA, SA tends to have relatively high computational cost as compared to tailored heuristic methods. Unlike the GA, an efficient method of parallelizing SA in a general way has yet to be demonstrated (Aarts and Korst 1989, Azencott 1992, Goldberg 1990, van Laarhoven and Aarts 1987, Mahfoud and Goldberg 1995, Rose *et al* 1990, Roussel-Ragot and Dreyfus 1990).

Indeed, this fact is stated directly by Aarts and Korst (1989, p 114), and later echoed by Goldberg (1990). The difficulty in parallelization arises from the fact that SA is essentially a sequential process. The typical SA algorithm can be viewed as a sequence of homogeneous *Markov chains* executing at monotonically decreasing temperature values, progressing from single solution to single solution. B2.2.2

Recently, researchers have been investigating hybrid algorithms that mix aspects of evolutionary and thermodynamic computational paradigms, such as GA and SA techniques. GA–SA hybrids are an attempt to provide an alternative general-purpose optimization technique that maintains the advantageous aspects of both paradigms while deemphasizing their respective drawbacks. Some of the earliest work done in the area of GA–SA hybrids is that of Sirag and Weisser (1987), Brown *et al* (1989), Lin *et al* (1991), Boseniuk and Ebeling (1991), and Mahfoud and Goldberg (1995).

Each of the above approaches is essentially a GA with varying degrees of coupling with SA operators. However, as stated by Mahfoud and Goldberg (1995), GA–SA hybrid designs should benefit from closer resemblance to SA than to the GA. This is a direct consequence of the increased convergence control offered by a SA temperature schedule. As expected, their approach resembles SA most closely of those listed above. It is a continuation of earlier work presented by Goldberg (1990) in which the incorporation of a population-oriented model within the SA paradigm is first suggested. The newly developed hybrid algorithm presented here also incorporates a population into traditional SA. However, there is a tighter coupling to the SA paradigm than in earlier hyrbids that allows for a more generalized parallelization regardless of the chosen temperature schedule. For this reason the new method is referred to as *population-oriented simulated annealing* (POSA) (Goldberg 1990, Varanelli and Cohoon 1995, Varanelli 1996).

G3.5.3.2 POSA

As mentioned above, the POSA heuristic is modeled after SA with efficient parallelization as a primary consideration. The parallel version of the POSA heuristic is modeled after the *parallel moves* approach to parallelizing SA (Aarts and Korst 1989, Azencott 1992, van Laarhoven and Aarts 1987, Roussel-Ragot and Dreyfus 1990). For a parallel moves approach, complete SA state transitions are performed concurrently with each processor contributing to the expected Boltzmann temporal distribution of solutions at the current SA temperature. This has been difficult to do efficiently in a general way using the standard Markovian SA model. For the POSA approach, we replace the single-state perturb–compare–accept/reject Metropolis SA model with the GA select–cross–mutate–evaluate model. A general outline of the parallel version of the POSA heuristic is

Par_POSA() {
  $k \leftarrow 0$; initialize($\mathcal{P}_k, t$); $a_{BSF} \leftarrow \min(\mathcal{P}_k)$;
  while (*stop criterion has not been met*) do
    while (*equilibrium has not been reached*) do
      for $i \leftarrow 1$ to $\frac{n}{2}$ do $\{a_{i1}, a_{i2}\} \leftarrow$ select($\mathcal{P}_k$);
      Concurrently for each of the $i \leftarrow 1$ to $\frac{n}{2}$ processors
      do $\{a'_{i1}, a'_{i2}\} \leftarrow$ crossover(a_{i1}, a_{i2});
        mutate(a'_{i1}); mutate(a'_{i2});
        evaluate(a'_{i1}); evaluate(a'_{i2});
        $\{a''_{i1}, a''_{i2}\} \leftarrow$ Metropolis_accept($t, a_{i1}, a_{i2}, a'_{i1}, a'_{i2}$);
      od
      $k \leftarrow k + 1$; $\mathcal{P}_k \leftarrow \emptyset$;
      for $i \leftarrow 1$ to $\frac{n}{2}$ do $\mathcal{P}_k \leftarrow \mathcal{P}_k \cup \{a'_{i1}, a'_{i2}\}$;
      if ($c(\min(\mathcal{P}_k)) < c(a_{BSF})$) then $a_{BSF} \leftarrow \min(\mathcal{P}_k)$;
    od
    decrement(t);
  od
  return(a_{BSF}) ; } .

For this algorithm we use the following notation. The set $\mathcal{P}_k$ is the kth population with $|\mathcal{P}_k| = n$. We consider t to be the annealing temperature, thus its decrement affects the *stop criterion*. The function $\min(\mathcal{P})$ returns the solution contained in $\mathcal{P}$ having the lowest cost, while $c(a)$ returns the cost of solution a. This algorithm is a fine-grained parallel version of POSA that might be suitable for a SIMD system with numerous processors. In particular, it considers there to be $n/2$ processors available for concurrent execution of crossover, mutation, and evaluation of solutions. In section G3.5.4 this is referred to as a *maximally centralized* algorithm.

It is important to note that the new state-space exploration strategy nullifies the asymptotic convergence

guarantee of standard SA since a near-Boltzmann temporal distribution cannot be guaranteed at each temperature. Goldberg (1990) showed that a near-Boltzmann temporal distribution can be achieved over a population at a given temperature with the use of a selection technique called *Boltzmann tournament selection* (BTS). Under this assumption, the use of BTS in the POSA heuristic would then carry over the asymptotic convergence guarantees of SA. However, BTS has several drawbacks limiting its practical application. Therefore, we forego the issue of asymptotic convergence, and instead focus on the empirical performance of the algorithm given different SA and GA design choices. The design choices for both the sequential and distributed POSA systems are described in the next section. The complete results for these systems are presented in section G3.5.4. C2.3

G3.5.3.3 Implementation details

There are a number of design decisions to be made for any GA or SA implementation. These decisions are equally important for GA–SA hybrids. They affect both the convergence behavior and the final solution quality exhibited by the algorithm. The chosen SA cooling schedule is based upon the classic Kirkpatrick–Gelatt–Vecchi schedule (1983). The schedule employs a constant temperature decrement rule of the form $t_k = t_0\alpha^k$. For the POSA implementation we set $\alpha = 0.95$. We set the initial temperature $t_0 = \sigma_\infty$, the standard deviation of the cost over the solution space (Aarts and Korst 1989, van Laarhoven and Aarts 1987) as estimated from a large sample of random solutions. The inner loop terminates when the number of matings is equal to the size of the SA neighborhoods as dictated by the chosen variation mechanism (Varanelli 1996). The algorithm terminates when the average cost over the population is found not to decrease over three consecutive iterations of the inner loop. For the purposes of this paper, we do not explore the effect of modifying these SA operators or the effect of switching to an adaptive cooling schedule.

For the GA operations, design choices must be made for the *population size*, *mating selection*, *crossover*, *mutation*, and *replacement* strategies. We choose a constant population size of 50 for the serial version of the heuristic, and sizes 80 and 160 for the distributed version. For mating selection, we use random pairings of population elements. If we use random selection, the host processor in a distributed or parallel implementation of the algorithm can carry out the selection procedure while the auxiliary processors are concurrently carrying out the crossover–mutate–evaluate–accept functions. If instead a fitness-based selection procedure is to be used, the host would have to wait for all of the auxiliary processors to finish their respective operations before the next mating selections could be made. E1.1, C2 C3.3, C3.2

An individual solution $\mathcal{P} = \{\mathcal{A}_\mathcal{P}, \mathcal{B}_\mathcal{P}\}$ is encoded as a bit vector with an element for each cell, that is, each element of $\mathcal{C}$. Since $\mathcal{P}$ is a bi-partition of $\mathcal{C}$, a 1 at vector position i can specify that $c_i \in \mathcal{A}_\mathcal{P}$ and a 0 can specify $c_i \in \mathcal{B}_\mathcal{P}$. Given this bit vector representation there are many possible choices for the crossover operator. The choice of crossover operator is generally based on the problem being solved. According to previously published results (Varanelli and Cohoon 1995, Varanelli 1996) *uniform crossover* is much more effective than either single- or two-point crossover early in a GA run for the VLSI-NPP, but is quickly affected by disruption (see also Spears and De Jong (1991) and Syswerda (1989)). On the other hand, single- and two-point crossover are slower to suffer from the effects of disruption, allowing continued improvement throughout the elongated run. For this reason, we use uniform crossover early in the POSA schedule, switching over to two-point crossover when the *best-so-far* (BSF) solution has not changed for 5 generations. Crossover is always performed with probability 1.0, due to the fact that the Metropolis acceptance criterion probabilistically allows the survival of a parent into the next generation. This is not the case in the simple GA, hence the lower crossover probability. C3.3.3.3

We use a rather standard bitwise mutation with probability, p_{mut}, in that for each individual the operator examines each bit position and with probability p_{mut} complements the value at that position. This mutation has the effect of randomly choosing to move each cell in an individual solution to the other partition element, for example, c_i moves from $\mathcal{A}_\mathcal{P}$ to $\mathcal{B}_\mathcal{P}$, with probability p_{mut}. In POSA we set $p_{mut} = 0.05$. As with the crossover probability, the mutation probability for the POSA heuristic is slightly higher than in the simple GA. This is an attempt to increase the exploration capability of the POSA heuristic. Again, this higher mutation rate is justified since the Metropolis criterion does not force the acceptance of all new offspring as in the simple GA. Other forms of mutation may be more beneficial. A greedy SA-like mutation may increase the local search capability while reducing the run times, but is not explored here.

The replacement strategy is based upon the SA Metropolis acceptance criterion (Kirkpatrick *et al*

1983). Each pairing of population elements results in two offspring. In an attempt to discourage a left- or right-end bias, each parent is randomly paired against one of the offspring for the Metropolis acceptance trial. If the randomly chosen child has a lower cost than the corresponding parent, the parent is automatically replaced in the population by the offspring. If the offspring has higher cost than the parent, the offspring replaces the parent probabilistically according to the Metropolis criterion involving the current temperature t. For POSA (as described in the pseudocodes on pages G3.5:4 and G3.5:8), the replacement strategy is performed by the Metropolis_accept function. That function takes as parameters the temperature, both parents, and both offspring, and returns two solutions according to the replacement strategy. The two returned solutions are then included in the next population $\mathcal{P}_k$.

G3.5.4 Performance results

Performance results are presented for both the sequential and distributed versions of the POSA VLSI-NPP heuristic. Both versions incorporate the GA and SA design choices outlined in the previous section. The sequential version is used to evaluate solution quality performance. The distributed version is used to demonstrate the efficiency and scalability of the parallelization of the method. The sequential version is implemented in the C++ programming language and executed on a four-CPU Sun SparcServer 10/51 with 256 Mbytes RAM. Results for the sequential version of the POSA VLSI-NPP heuristic are generated using the PrimarySC1 (833 cells, 904 networks) and PrimarySC2 (3014 cells, 3029 networks) SIGDA standard cell benchmark circuits (Preas 1987), plus two randomly generated instances, rand.100 (100 cells, 100 networks) and rand.250 (250 cells, 250 networks). The random circuits are generated in such a manner as to have similar network size and interconnectivity distributions as the benchmark circuits.

The distributed version is implemented under the Mentat distributed computing environment (Grimshaw 1993), using C++ as the basis for its command language. The execution platform for the distributed version is a heterogeneous network of ten Sun workstations of varying CPU and RAM configurations, ranging from a single-CPU SparcStation 2 with 40 Mbytes RAM to a two-CPU SparcStation 20 with 256 Mbytes RAM. The results for the distributed version are compared to the sequential version running on the most powerful machine in the network. The PrimarySC1 (PrSC1) and PrimarySC2 (PrSC2) benchmarks are used as the test instances.

G3.5.4.1 Solution quality

For the VLSI-NPP the average-case and best-case solution quality of the sequential version of the POSA heuristic is compared with three other heuristic methods: FM, an implementation of the algorithm of Fiduccia and Mattheyses (1982), Classic SA, an implementation of the simulated annealing of Kirkpatrick *et al* (1983) using the SA parameter settings discussed in the previous section, and Simple GA, an implementation of the GA of Holland (1975). These three methods are used for the comparison due to their 'superior solution quality' (Johnson *et al* 1989).

For these comparisons we want to compare the quality of the solutions derived after comparable amounts of computation. However, for FM, Classic SA, and POSA, each run has a dynamic termination criterion, making 'execute for XX CPU seconds' an inappropriate specification for a single run. Rather, we select a total time, TOTtime, and allow each algorithm to execute a sequence of individual independent runs until the total elapsed CPU time is TOTtime (or near to TOTtime). Thus, the number of runs in a sequence varies for each algorithm but the computation time to execute each entire sequence of runs is comparable. The results are then averaged over the results from each run in a sequence. For this study, TOTtime is selected to be the CPU time required to complete a sequence of five runs of the Simple GA, with each run comprising 20 000 generations. The results are presented in tables G3.5.1 and G3.5.2. As can be seen in the tables, the POSA VLSI-NPP heuristic compares favorably with the other three methods when given equal amounts of CPU time.

Table G3.5.1 shows that POSA easily outperforms the Simple GA in both average and best-case solution quality for all instances. In table G3.5.2 one can see that POSA outperforms FM with respect to average solution quality for three of the four instances and with respect to best-case solution quality for two of the four instances. POSA showed performance equal or superior to that of Classic SA with respect to both average and best-case solution quality for two of the instances. Classic SA outperformed POSA by a very small percentage in both average and best-case solution quality (1.6 and 0.8%, respectively) for the instance rand.250.

Table G3.5.1. Experimental results on four VLSI-NPP instances for comparing the POSA heuristic with Simple GA, given comparable amounts of computation time.

	Simple GA					POSA w/bitwise mutation				
Problem instance	Avg. cut	Best cut	Avg. generation count	Runs	Avg. CPU (s)	Avg. cut	Best cut	Avg. generation count	Runs	Avg. CPU (s)
rand.100	**21**	**18**	20 000	5	431	**17**	**14**	168	250	9.5
rand.250	**98**	**86**	20 000	5	1 373	**65**	**55**	460	125	46
PrSC1	**119**	**103**	20 000	5	3 780	**98**	**78**	912	50	299
PrSC2	**1129**	**1095**	20 000	5	14 550	**643**	**582**	5764	50	1273

Table G3.5.2. Experimental results on the same instances as table G3.5.1 for comparing POSA with the VLSI-NPP heuristics, FM and Classic SA. Again, each method is given comparable amounts of computation time on each instance.

	FM				Classic SA			
Problem instance	Avg. cut	Best cut	Runs	Avg. CPU (s)	Avg. cut	Best cut	Runs	Avg. CPU (s)
rand.100	**19**	**16**	20 000	0.1	**17**	**14**	20 000	0.1
rand.250	**68**	**58**	35 000	0.2	**61**	**53**	15 000	0.4
PrSC1	**127**	**72**	10 000	1.7	**116**	**85**	2500	6.6
PrSC2	**460**	**325**	5 000	13.3	**381**	**305**	2500	25.0

However, it should be noted that both Classic SA and FM are able to outperform POSA in terms of average and best-case solution quality for the PrimarySC2 instance (PrSC2). This is not unexpected due to the relatively poor performance of the Simple GA on this instance, combined with the lack of problem-specific fine tuning for either the SA or GA parameters for the VLSI-NPP. The poor performance of the Simple GA for the PrimarySC2 instance can be traced to the fact that the VLSI-NPP is a GA-loose problem, implying that it is prone to the effects of disruption possibly causing premature convergence due to a rapid homogenization of the population. This is compounded by the fact that the PrimarySC2 instance is very large, and many of the specified hyperplanes have defining lengths approaching the entire length of the population strings.

As table G3.5.1 indicates, there is a considerable improvement in both average and best-case solution quality performance seen in the POSA heuristic as compared to the Simple GA for all four problem instances. In particular, POSA is able to converge to a significantly better solution than the Simple GA in a fraction of the total number of generations for these instances. This indicates that fine tuning both the SA cooling schedule and the GA parameters for the specific problem being examined should bring solution quality more in line with other methods.

G3.5.4.2 Speedup through parallelization

A distributed prototype of the POSA algorithm described in the previous subsections has been developed in order to demonstrate the efficacy of a parallel version of the algorithm. As was previously discussed, the distributed prototype is implemented under the Mentat distributed computing system (Grimshaw 1993) with a network of workstations serving as the execution platform. In light of this particular computing architecture, a method of distributing the GA population strings amongst the available processors with minimum communications overhead is needed.

The population distribution method commonly used by researchers utilizing GAs on distributed architectures is some sort of *subspeciation* technique, in which each processor independently evolves a *subpopulation* of strings (Bianchini and Brown 1993, Cohoon *et al* 1991, Davis 1987). Since larger populations tend to produce better GA solution quality through improved diversity, all of these population distribution schemes involve some mechanism for the migration of solutions between processors. This

allows for a rediversification of solutions in the subpopulations which can quickly exhibit solution homogeneity due to their relatively small size. This is the approach taken in the distributed version of the POSA heuristic.

An important aspect in the distribution of the population is the amount of *centralization* employed (Bianchini and Brown 1993). A maximally centralized distributed GA has the entire population managed by a master processor, simply sending pairs of solutions to each of the auxiliary processors for the application of the genetic operators. A minimally centralized distributed GA is one in which the population is evenly distributed amongst the available processors with no migration of solutions between processors during evolution. Bianchini and Brown (1993) present a study of distributed GAs finding that a more centralized population can produce better quality solutions than a more distributed population given an equal number of GA generations. However, the trade off is clear. Greater centralization requires greater communications overhead, hence greater computation times in light of a specified number of generations. As a result, most distributed GAs fall somewhere between maximally centralized and minimally centralized.

The results shown in table G3.5.3 are for a distributed version of the POSA heuristic we have implemented using a medium-centralized configuration as described in the following outline of the parallel POSA heuristic, where t is the annealing temperature, n is the total population size and m is the number of processors:

Par_POSA() {
  $k \leftarrow 0$; initialize($\mathcal{P}_k, t$); $a_{BSF} \leftarrow \min(\mathcal{P}_k)$;
  while (*stop criterion has not been met*) do
    $\{\mathcal{P}_{10}, \ldots, \mathcal{P}_{m0}\} \leftarrow$ Partition($\mathcal{P}_k$)
    Concurrently for each of the $i \leftarrow 1$ to m processors
    do $j \leftarrow 0$; psize $\leftarrow |\mathcal{P}_{ij}|$;
      while (*equilibrium has not been reached*) do
        for $p \leftarrow 1$ to $\frac{\text{psize}}{2}$ do
          $\{a_{ip1}, a_{ip2}\} \leftarrow$ select($\mathcal{P}_{ij}$);
          $\{a'_{ip1}, a'_{ip2}\} \leftarrow$ crossover(a_{ip1}, a_{ip2});
          mutate(a'_{ip1}); mutate(a'_{ip2});
          evaluate(a'_{ip1}); evaluate(a'_{ip2});
          $\{a''_{ip1}, a''_{ip2}\} \leftarrow$ Metropolis_accept($t, a_{ip1}, a_{ip2}, a'_{ip1}, a'_{ip2}$);
        od
        $j \leftarrow j + 1$; $\mathcal{P}_{ij} \leftarrow \emptyset$;
        for $p \leftarrow 1$ to $\frac{\text{psize}}{2}$ do $\mathcal{P}_{ij} \leftarrow \mathcal{P}_{ij} \cup \{a''_{ip1}, a''_{ip2}\}$;
        if ($c($ min($\mathcal{P}_{ij}$) $) < c(a_{BSF})$) then $a_{BSF} \leftarrow \min(\mathcal{P}_{ij})$;
      od
    od
    $k \leftarrow k + 1$; $\mathcal{P}_k \leftarrow$ Collect($\mathcal{P}_{1j}, \ldots, \mathcal{P}_{mj}$);
    decrement(t);
  od
  return(a_{BSF}) ; }

Metropolis_accept(t, a_1, a_2, a'_1, a'_2) {
  if (rand(0, 1) < 0.5) then { $b_1 \leftarrow$ Choose(t, a_1, a'_1)
                                   $b_2 \leftarrow$ Choose(t, a_2, a'_2) }
    else { $b_1 \leftarrow$ Choose(t, a_1, a'_2)
           $b_2 \leftarrow$ Choose(t, a_2, a'_1) } ;
  return({ b_1, b_2 }) ; }

Choose(t, a, a') {
  if ($c(a') < c(a)$) then $b \leftarrow a'$
    else if (rand(0, 1) $\leq \exp(\frac{c(a')-c(a)}{t})$) then $b \leftarrow a'$
      else $b \leftarrow a$;
  return(b) ; }

A master processor administers the SA schedule, as well as maintaining its own subpopulation. The auxiliary processors simply perform the specified GA operations on their respective subpopulations. The subpopulation statistics are sent back to the master processor to aid in SA schedule administration. In the actual implementation, rediversification of the subpopulations is performed at the end of each SA temperature by the master randomly pairing the processors, with each pair exchanging a randomly chosen 5% of their populations.

Table G3.5.3. Results for the distributed version of POSA.

	PrimarySC1				PrimarySC2			
	Population = 80		Population = 160		Population = 80		Population = 160	
Number of processors	Avg. cut	Avg. T_{WC} (s)	Avg. cut	Avg. T_{WC} (s)	Avg. cut	Avg. T_{WC} (s)	Avg. cut	Avg. T_{WC} (s)
1	**100**	683	**96**	1276	**572**	3074	**534**	5931
2	**98**	466	**96**	803	**560**	1952	**521**	3589
4	**100**	376	**93**	551	**568**	1338	**521**	2078
8	**103**	308	**97**	393	**571**	841	**524**	1479
10	**102**	296	**100**	360	**572**	795	**531**	1345

The results presented in table G3.5.3 are averaged over ten runs, with T_{WC} representing the wall clock time. As can be seen in the table, solution quality is approximately the same for both the serial and distributed versions in the case of PrimarySC1. This has very much to do with the fact that the serial version is already producing solutions of exceptional quality for this instance. There is little room for improvement. However, in the case of PrimarySC2, there is a slight improvement in solution quality seen in the distributed version over the serial version. As discussed in the previous subsection, the PrimarySC2 instance is highly susceptible to disruption. This problem is overcome more successfully in the distributed version due to the rediversification of population elements at the end of each SA temperature (although this effect is seen to lessen as the subpopulation sizes decrease with increasing number of processors).

The results shown in table G3.5.3 require some clarification. One specific aspect of the above distributed test scenario that needs further examination is the effect of network traffic and CPU utilization on the results presented in table G3.5.3. Both the physical network and the CPUs used in producing the above data are shared resources in a multiuser environment. What are the results if the communications network and CPUs are unshared resources? Another aspect of the test scenario that has not been taken into account is the nondeterminism of the algorithm itself. What are the results if both the initial and final POSA temperatures are kept constant for each instance, ensuring that each run is performing exactly the same number of operations?

These two questions are addressed by the results presented in table G3.5.4. The results for the PrimarySC1 and PrimarySC2 instances presented in table G3.5.4 were generated using a dedicated network of eight Sun SparcStation 2s, each with 28 Mbytes of RAM. The distributed POSA processes are allowed to consume 100% of each corresponding CPU's compute cycles, with no competing processes allowed at the time of the trials. For the PrimarySC1 instance, initial and final POSA temperatures are kept constant at $t_0 = 13.0$ and $t_f = 0.01$, while these values were set to $t_0 = 25.0$ and $t_f = 0.01$, respectively, for the PrimarySC2 instance. These values were chosen empirically as typical for the two given instances. The resulting speedup figures for the dedicated network are compared to those utilizing the shared network, up to eight processors. As can be seen in table G3.5.4, speedup figures for the shared and unshared networks are quite similar, with the speedup seen in the unshared network slightly better than the speedup seen in the shared network. These small differences are probably attributable to the lower network traffic and higher CPU utilization in the dedicated tests.

The speedup seen in the distributed version of POSA over the serial version as presented in table G3.5.4 is illustrated graphically in figure G3.5.2. As can be seen in the figure, appreciable speedup is noted over the serial version for an increasing number of processors up to ten. As can be seen in the graphs, however, the speedup is far from linear. This can be attributed to a number of factors, including CPU and network utilization loads, as well as overhead inherent to the Mentat system. It should be noted, however, that speedup is still increasing at the upper limit of each graph, indicating further potential scalability with

Table G3.5.4. A comparison of distributed POSA on the VLSI-NPP for a fixed number of generations utilizing shared and unshared network resources.

	PrimarySC1				PrimarySC2			
	Population = 80		Population = 160		Population = 80		Population = 160	
No of proc.	Speedup shared	Speedup unshared	Speedup shared	Speedup unshared	Speedup shared	Speedup unshared	Speedup shared	Speedup unshared
1	1	1	1	1	1	1	1	1
2	1.47	1.58	1.59	1.77	1.57	1.69	1.65	1.83
4	1.82	1.89	2.32	2.44	2.30	2.41	2.85	2.99
8	2.22	2.26	3.25	3.35	3.66	3.75	4.01	4.15

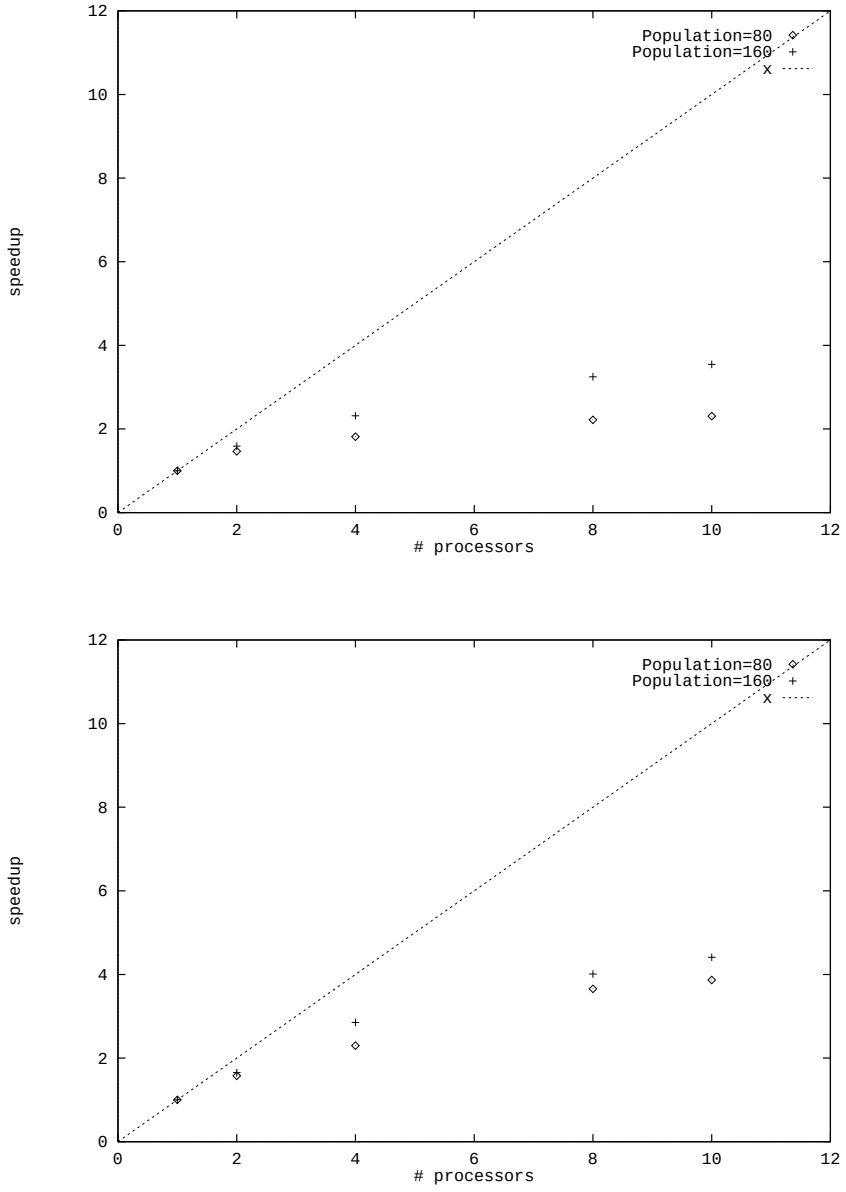


Figure G3.5.2. Speedup curves for the distributed version of POSA on VLSI-NPP instances PrimarySC1 (top) and PrimarySC2 (bottom).

a larger number of processors. The speedup graph for PrimarySC1 is seen to flatten more quickly than that of PrimarySC2, due to the lower computation/communication ratio inherent to this instance.

These data indicate the feasibility of a parallel implementation, assuming the existence of an efficient message-passing mechanism for the target architecture. This is directly attributable to the fact that the coarse-grain parallelization exploited by the proposed POSA methodology has an inherently high computation/communications ratio, especially when applied to the VLSI-NPP. Indeed, it was shown by Varanelli (1996) that, for the given VLSI-NPP example, the computation times at each processor are increasing at a rate of $O(n^3)$ while communication times are increasing at an $O(n)$ rate, where n is the size of the input instance. Even in a distributed environment with a shared communications network and shared CPU resources such as with the Mentat implementation of POSA described previously, the total computation time far outweighs total communication time. Hence, the efficacy of a parallel implementation on a dedicated parallel architecture with an unshared communications network and efficient message-passing mechanism is indeed illustrated.

G3.5.5 Conclusion

We have presented an evolutionary–thermodynamic hybrid technique called POSA for the VLSI network partitioning problem. The algorithm combines the advantages of both the GA and SA paradigms. The GA's population-oriented model of computation allows for efficient parallelization that is not possible with standard SA. The SA cooling schedule gives the user greater convergence control than is possible with traditional GA operators alone. Additionally, the operators allow for fine tuning of the heuristic to many different types of problem to be solved. Experimental results indicate that the algorithm is capable of outperforming other popular VLSI-NPP heuristic methods given equal amounts of CPU time. Results also indicate that the method is efficiently parallelizable and scalable in nature.

Acknowledgements

This work was supported in part by the National Science Foundation through grants MIP9107717 and CCR9224789. This support is greatly appreciated.

References

Aarts E H L and Korst J H M 1989 *Simulated Annealing and Boltzmann Machines: a Stochastic Approach to Combinatorial Optimization and Neural Computing* (Chichester: Wiley)

Azencott R (ed) 1992 *Simulated Annealing: Parallelization Techniques* (New York: Wiley)

Bianchini R and Brown C M 1993 Parallel genetic algorithms on distributed-memory architectures *Proc. 6th Conf. N. Am. Transputer Users Group (Vancouver)* pp 67–82

Boseniuk T and Ebeling W 1991 Boltzmann-, Darwin-, and Haeckel-strategies in optimization problems *Proc. 1st Conf. on Parallel Problem Solving from Nature (Dortmund, 1990)*) (*Lecture Notes in Computer Science 496*) ed H-P Schwefel and R Männer (Berlin: Springer) pp 430–44

Brown D E, Huntley C L and Spillane A R 1989 A parallel genetic heuristic for the quadratic assignment problem *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, June 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 406–15

Cohoon J P, Hegde S U, Martin W N and Richards D S 1991 Distributed genetic algorithms for the floorplan design problem *IEEE Trans. Comput.-Aided Design Integ. Circuits Syst.* **CADICS-10** 483–92

Davis L (ed) 1987 *Genetic Algorithms and Simulated Annealing* (London: Pitman)

Dunlop A E and Kernighan B W 1985 A procedure for placement of standard-cell VLSI circuits *IEEE Trans. Comput.-Aided Design Integ. Circuits Syst.* **CADICS-4** 92–8

Fiduccia C M and Mattheyses R M 1982 A linear-time heuristic for improving network partitions *Proc. 19th ACM/IEEE Design Automation Conference (Las Vegas, NV)* pp 241–7

Garey M R and Johnson D S 1979 *Computers and Intractability: a Guide to the Theory of NP-Completeness* (San Francisco, CA: Freeman)

Goldberg D E 1990 A note on Boltzmann tournament selection for genetic algorithms and population-oriented simulated annealing *Complex Syst.* **4** 445–60

Grimshaw A S 1993 Easy to use object-oriented parallel programming with Mentat *IEEE Comput.* **26** 39–51

Holland J H 1975 *Adaptation in Natural and Artificial Systems* (Ann Arbor, MI: University of Michigan Press)

Johnson D S Aragon C R McGeoch L A and Schevon C 1989 Optimization by simulated annealing: an experimental evaluation; Part 1, Graph partitioning *Operat. Res.* **37** 865–892

Kernighan B W and Lin S 1970 An efficient heuristic for partitioning graphs *Bell Syst. Tech. J.* **49** 291–307

Kirkpatrick S, Gelatt C D and Vecchi M P 1983 Optimization by simulated annealing *Science* **220** 45–54

Lin F-T, Kao C-Y and Hsu C-C 1991 Incorporating genetic algorithms into simulated annealing *Proc. 4th Int Symp. on Artificial Intelligence* pp 290–7

Mahfoud S W and Goldberg D E 1995 Parallel recombinative simulated annealing: a genetic algorithm *Parallel Comput.* **21** 1–28

Preas B 1987 Benchmarks for cell-based layout systems *Proc. 24th ACM/IEEE Design Automation Conference (Miami Beach, FL)* pp 319–20

Rose J S, Snelgrove W M and Vranesic Z G 1990 Parallel standard cell placement algorithms with quality equivalent to simulated annealing *IEEE Trans. Comput.-Aided Design Integ. Circuits Syst.* **CADICS-9** 253–9

Roussel-Ragot P and Dreyfus G 1990 A problem-independent parallel implementation of simulated annealing: models and experiments *IEEE Trans. Comput.-Aided Design Integ. Circuits Syst.* **CADICS-9** 827–35

Sirag D J and Weisser P T 1987 Toward a unified thermodynamic genetic operator *Proc. 2nd Int. Conf. on Genetic Algorithms (Cambridge, MA, 1987)* ed J J Grefenstette (Hillsdale, NJ: Erlbaum) pp 116–22

Spears W M and De Jong K A 1991 On the virtues of parameterized uniform crossover *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, July 1991)* ed R Belew and L Booker (San Mateo, CA: Morgan Kaufmann) pp 230–6

Syswerda G 1989 Uniform crossover in genetic algorithms *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, June 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 2–9

Varanelli J M 1996 *On the Acceleration of Simulated Annealing* PhD Dissertation, Department of Computer Science, University of Virginia

Varanelli J M and Cohoon J P 1995 Population-oriented simulated annealing: a genetic/thermodynamic hybrid approach to optimization *Proc. 6th Int. Conf. on Genetic Algorithms (Pittsburgh, PA, July 1995)* ed L J Eshelman (San Mateo, CA: Morgan Kaufmann)

van Laarhoven P J M and Aarts E H L 1987 *Simulated Annealing: Theory and Applications* (Dordrecht: Reidel)

G3.6 The Evolutionary Planner/Navigator in a mobile robot environment

Jing Xiao

Abstract

Based on evolutionary computation concepts, the Evolutionary Planner/Navigator (EP/N) represents a new approach to path planning and navigation. The major advantages of the EP/N include being able to achieve both near-optimality of paths and high planning efficiency, being able to accommodate different optimization criteria, being flexible to changes, and being robust to uncertainties. The EP/N unifies off-line planning and on-line planning/navigation processes in the same evolutionary algorithm to deal with unknowns in an environment gracefully and flexibly. It provides high safety for the robot without requiring complete information about the environment.

G3.6.1 Project overview

The *motion planning* problem for mobile robots is typically formulated as follows (Yap 1987): given a robot and a description of an environment, plan a path of the robot between two specified locations which is collision free and satisfies certain optimization criteria. Traditionally there are two approaches to the problem: off-line planning, which assumes a perfectly known and stable environment, and on-line planning, which focuses on dealing with uncertainties when the robot traverses the environment. On-line planning is also referred to by many researchers as the *navigation* problem. (Although some researchers also interpret *navigation* as a low-level control problem for path following, we do not use such an interpretation here.)

A great deal of research has been done in motion planning and navigation (see Yap 1987 and Latombe 1991 for surveys). However, different existing methods encounter one or many of the following difficulties:

- high computation expenses
- inflexibility in responding to changes in the environment
- inflexibility in responding to different optimization goals
- inflexibility in responding to uncertainties
- inability to combine advantages of global planning and reactive planning.

The EP/N system was developed to address these difficulties; the inspiration to use evolutionary techniques was triggered by the following ideas/observations:

- Randomized search can be the most effective in dealing with NP-hard problems and in escaping local minima.
- Parallel search actions not only provide great speed but also provide ground for *interactions* among search actions to achieve even greater efficiency in optimization.
- Creative application of the evolutionary computation *concept* rather than dogmatic imposition of a standard algorithm proves to be more effective in solving specific types of real problems.
- Intelligent behavior is the result of a collection of simple reactions to a complex world.
- A planner can be greatly simplified, much more efficient and flexible, and increase the quality of search, if search is not confined to be within a specific map structure.
- It is more meaningful to equip a planner with the flexibility of changing the optimization goals than the ability of finding the absolutely optimum solution for a single, particular goal.

The EP/N embodies the above ideas by following the evolution program approach, that is, combining the concept of evolutionary computation with problem specific chromosome structures and genetic operators (Michalewicz 1994). With such an approach, the EP/N pursues all the advantages as described above. Less obvious, though, is that, with the unique design of chromosome structure and genetic operators, the EP/N does not need a discretized map for search, which is usually required by other planners. Instead, the EP/N 'searches' the original and continuous environment by generating paths based on evolutionary computation. The objects in the environment can simply be indicated as a collection of straight-line 'walls'. This representation accommodates both known objects and partial information of unknown objects obtained from sensing. Thus, there is little difference between off-line planning and on-line navigation for the EP/N. In fact, the EP/N unifies off-line planning and on-line navigation in the same evolutionary algorithm and chromosome structure.

The structure of the EP/N is shown in figure G3.6.1, where FEG—the off-line evolutionary algorithm, and NEG—the on-line evolutionary algorithm—are essentially the same evolutionary algorithm as to be described. The only difference between FEG and NEG is in certain values of parameters (see section G3.6.5) one may choose. The different parameter values are to accommodate slightly different objectives of FEG and NEG: FEG emphasizes the optimality of a path while NEG emphasizes the swiftness in generating a feasible path. Note that both FEG and NEG perform global planning, and NEG generates an alternative subpath by global planning based on the updated knowledge of the environment obtained from sensing. Moreover, if no object is initially known in the environment, then FEG will generate a straight-line path with just two nodes: the start and the goal locations. It will solely depend on the NEG to lead the robot towards the goal while avoiding unknown or newly emerged obstacles.

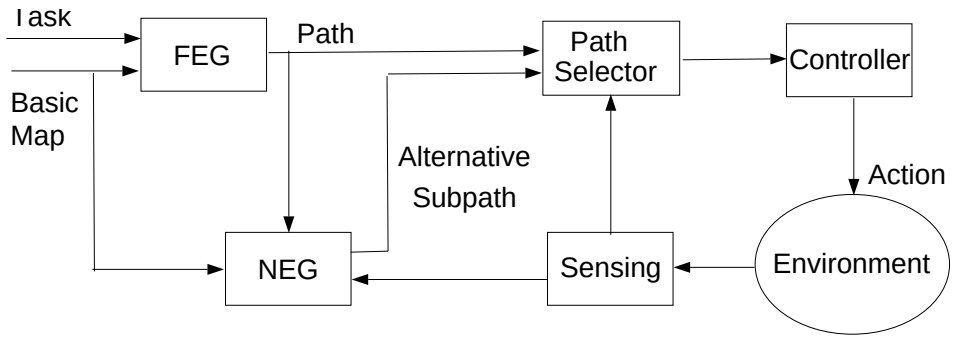


Figure G3.6.1. The EP/N structure.

G3.6.2 Design process

We now describe the evolutionary algorithm which both FEG and NEG adopt in detail.

G3.6.2.1 Chromosomes and initialization

A path consists of one or more straight-line segments, with the starting location, the goal location, and (possibly) the intersection locations of two adjacent segments defining the *nodes*. A feasible path consists of only feasible nodes. An infeasible path has at least one infeasible node which is either not connectable to the next node on the path due to obstacles or located inside some obstacle.

Chromosomes are represented as ordered lists of path nodes: each node, apart from the pointer to the next node, consists of x and y coordinates of the knot point and a state variable b, which indicates whether or not the node is feasible (figure G3.6.2). Each chromosome can have a varied number of nodes, which provides great flexibility. The methods for checking the feasibility of a node (i.e. location validity and connectivity) are relatively simple and are based on algorithms described by Pavlidis (1982).

The initialization of chromosomes is a random process subject to the following input parameters: a population size P and the maximum number of nodes in a chromosome N. For each chromosome, a random number is generated within $[2, N]$ to determine its length, that is, the number of nodes. The coordinates x and y are also created randomly for each node of such a chromosome within the confine of the environment. P chromosomes are generated in this way.

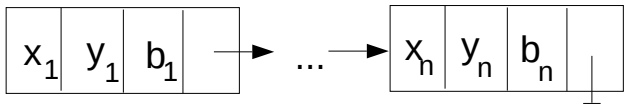


Figure G3.6.2. A chromosome representing a path.

G3.6.2.2 Evaluation

The evaluation function Path_Cost(p) measures the path cost of a chromosome p. Since p can be either feasible or infeasible, we adopt two separate evaluation functions eval_f and eval_u to handle the feasible and infeasible cases respectively. Our design of the evaluation function Path_Cost(p) has gone through a long process of development, as will be discussed in section 3.6.3. The eval_f and eval_u to be described are the most recent results of such development.

It seems to be relatively easy to compare two feasible paths. Intuitively, we think eval_f should be a function of the total length of a path dist, its smoothness smooth and the clearance clear between the path and the surrounding obstacles. There can be many ways to define the function eval_f. At present, we simply define it as the linear combination of dist, smooth, and clear:

$$\text{eval}_\text{f}(p) = w_\text{d}\,\text{dist}(p) + w_\text{s}\,\text{smooth}(p) + w_\text{c}\,\text{clear}(p)$$

where the constants w_d, w_s, and w_c represent the weights on the total cost of the path's length, smoothness, and clearance, respectively. We define dist, smooth, and clear as the following:

- $\text{dist}(p) = \sum_{i=1}^{n-1} d(m_i, m_{i+1})$, the total length of the path, where $d(m_i, m_{i+1})$ denotes the distance between two adjacent path nodes m_i and m_{i+1}.
- $\text{smooth}(p) = \max_{i=2}^{n-1} s(m_i)$, the maximum 'curvature' at a knot point, where 'curvature' is defined as
$$s(m_i) = \frac{\theta_i}{\min\{d(m_{i-1}, m_i), d(m_i, m_{i+1})\}}$$
and $\theta_i \in [0, \pi]$ is the angle between the extension of the line segment connecting nodes m_{i-1} and m_i and the line segment connecting nodes m_i and m_{i+1} (figure G3.6.3).
- $\text{clear}(p) = \max_{i=1}^{n-1} c_i$, where
$$c_i = \begin{cases} g_i - \tau & \text{if } g_i \geq \tau \\ \text{e}^{a(\tau - g_i)} - 1 & \text{otherwise} \end{cases}$$
g_i is the smallest distance from the segment $\overline{m_i m_{i+1}}$ to all detected objects, τ is a parameter defining a 'safe' distance, and a is a coefficient.

With this formulation, our goal is to minimize the function eval_f.

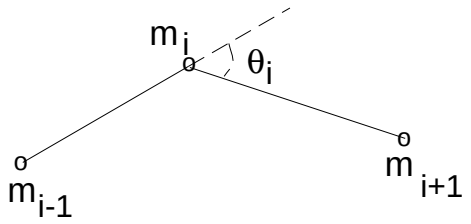


Figure G3.6.3. θ_i at each node m_i.

We took into account several factors in the design of eval_u: the number of intersections of a path with obstacles, the depth of intersection (i.e. how deeply a path cuts through obstacles), the ratio between the numbers of feasible and infeasible segments, the total lengths of feasible and infeasible segments, and so on, and implemented two designs for eval_u.

One design of eval_u is as follows:

$$\text{eval}_\text{u}(q) = \mu + \eta$$

where μ is the number of intersections of a whole path with obstacles and η is the average number of intersections per infeasible segment. With this evaluation function, the path costs of the three paths from figure G3.6.4 are

$$\begin{aligned}\text{eval}_{\text{u}}(p) &= 2 + 2 = 4\\ \text{eval}_{\text{u}}(q) &= 4 + 2 = 6\\ \text{eval}_{\text{u}}(r) &= 4 + 4 = 8\end{aligned}$$

which match our intuition: path p is the one which will generate a feasible offspring most easily, and path q is much more promising than path r. This eval_{u}, however, may not be perfect, since q could be considered the best among the three paths (i.e. it should have the lowest cost) from a different perspective.

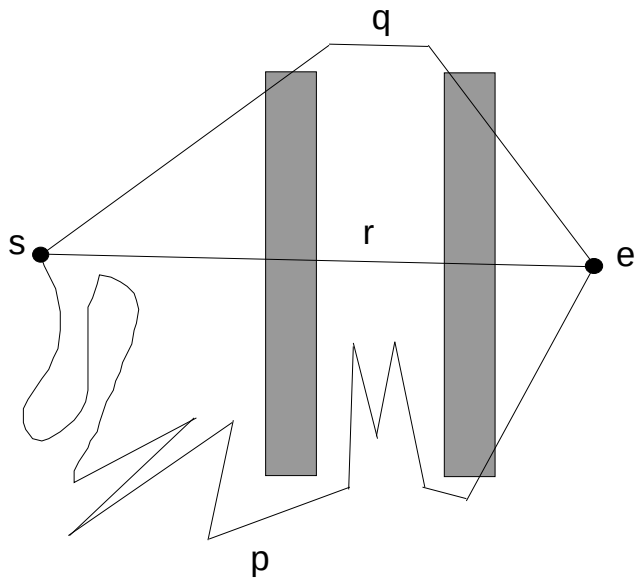


Figure G3.6.4. Three infeasible paths p, q, and r.

The other design makes eval_{u} equal to the summation of all the penetration distances, where a penetration distance D is defined as the minimum distance to move an infeasible path segment out of an obstacle it penetrates. This design is reasonable in almost all cases but is more computationally expensive than the first design.

Associated with this approach of designing eval_{u} independently of eval_{f} is the issue of how to compare feasible paths against infeasible ones. This issue requires the answer of the following question:

Is *any* feasible solution better than *any* infeasible one?

In the EP/N, we have chosen the (somewhat risky) answer 'yes', which makes such comparisons relatively easy for us and is also consistent with our designs of eval_{u}. With this choice, we add to the value of eval_{u} of any infeasible path p a constant ρ (within a given generation of the evolutionary process) to make the path less attractive than a feasible one:

$$\rho = \max\{0, \max_{p \in F}\{\text{eval}_{\text{f}}(p)\} - \min_{q \in U}\{\text{eval}_{\text{u}}(q)\}\}$$

where F and U denote the sets of feasible and infeasible paths respectively. Note that ρ measures the difference between the worst feasible and the best infeasible paths.

In actual implementation, we do not really compute ρ; instead, we simply sort the feasible paths and infeasible paths separately from the best to the worst based on their separate evaluation functions. Then, we 'append' the sorted list of infeasible paths at the tail of the sorted list of feasible paths. (This works with any ranking selection.)

G3.6.2.3 Genetic operators

The current version of EP/N uses eight types of genetic operator to evolve chromosomes into possibly better ones. These operators are sufficient to generate an *arbitrary* path, but may not all be needed in all situations. The application of each operator is controlled by a probability. How to select the best combination of operators, that is, how to determine those probabilities, very much depends on environmental characteristics and specific constraints imposed on a task. Our current version of EP/N is able to feed back how useful an operator is, which helps us in determining the probabilities. However, more research is needed (see G3.7.5). From our current experience on fairly complex environments, the EP/N system performed the best with all eight types of operator present with considerable probabilities (e.g. in the range 0.5–0.9).

Now we introduce each type of operator, as illustrated in figure G3.6.5:

crossover: recombines two (parent) paths into two new paths. The parent paths are divided randomly into two parts respectively and recombined: the first part of the first path with the second part of the second path, and the first part of the second path with the second part of the first path. Note that there can be different numbers of nodes in the two parent paths.

mutation_1: used for fine tuning node coordinates in a path for shape adjustment.

mutation_2: used for large change of node coordinates in a path.

insertion: inserts new nodes into a path.

deletion: deletes nodes from a path.

swap: swaps the coordinates of selected adjacent nodes in a path.

smooth: smooths turns of a feasible path by 'cutting corners', that is, for a selected node, the operator inserts two new nodes on the two path segments connected to that node respectively and deletes that selected node.

repair: repairs an infeasible segment in a path by 'pulling' the segment around its intersecting obstacles.

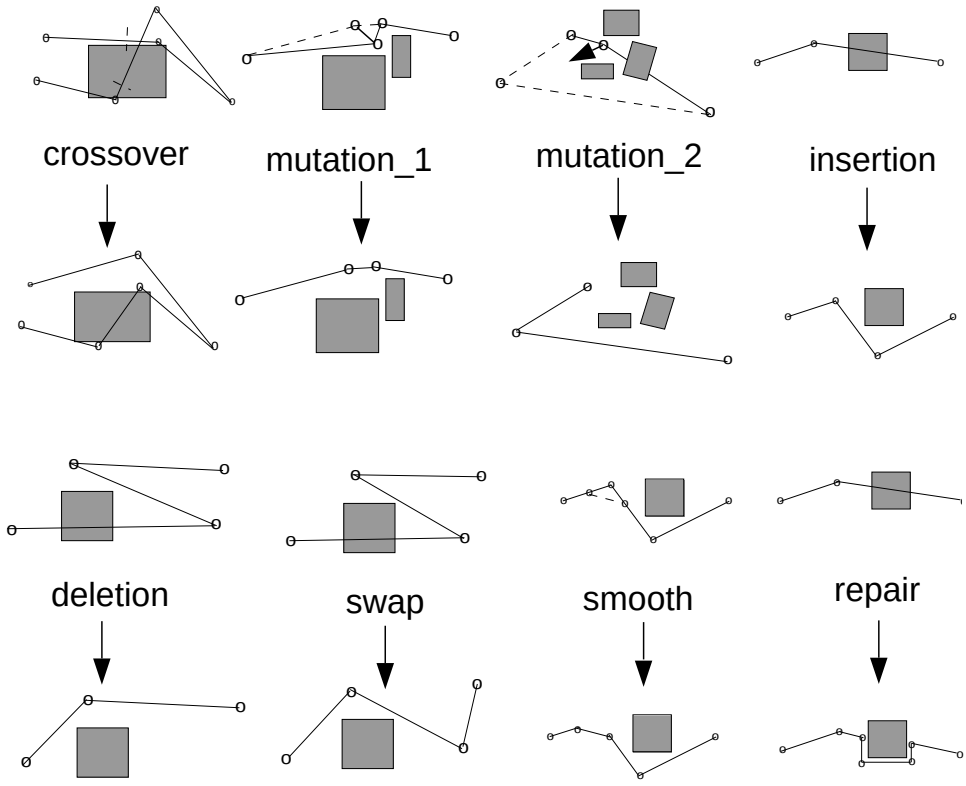


Figure G3.6.5. The roles of the genetic operators.

Note that we deliberately left out details on how exactly nodes were *selected* and *changed* in many operators, since such decisions could be made in various ways from purely random to incorporating much heuristic knowledge. In the earlier EP/N, such decisions were made mostly randomly. The current EP/N is equipped with versions of operators using more knowledge. For example, it has two versions of **mutation_1**. The first version changes the coordinates of a node randomly within some bounds which decrease as evolution proceeds; it applies to any path. The second version, however, applies to only a feasible path, and it changes the coordinates of a (feasible) node randomly within some local clearance of the path so that the path remains feasible afterwards. Both versions select nodes randomly. The merits of different types and versions of operator will be further discussed in G3.6.3.

G3.6.2.4 Reproduction

For the selection process, a population of P chromosomes are first sorted based on their fitness values (i.e. cost values) from the best to the worst, and a *roulette wheel* of P slots is then produced with the ith slot sized proportional to the fitness value of the ith chromosome (Michalewicz 1994; also see Section C2.2). By 'spinning' the wheel, the chromosomes which have better fitness values (i.e. lower cost values) will have better chances to be selected for reproduction. C2.2

In order to be more efficient, in a later version of the EP/N, we adopted a fixed roulette wheel of P slots with linearly decreasing slot sizes instead of generating a different roulette wheel at each generation. In this way, the chance for a chromosome to be selected is not necessarily proportional to its fitness value but is still better than the chance for a worse chromosome to be selected.

Generally, a parameter $S \leq P$ determines the number of chromosomes to be selected for reproduction. At generation t, the selected S chromosomes from the population $P(t)$ are altered by the genetic operators to generate S offspring. The S offspring plus the $P - S$ best chromosomes in the original population $P(t)$ form the next generation of population $P(t+1)$.

In our latest version of EP/N, only one genetic operator is used at each generation, and $S = 1$ (or 2 if the operator chosen is **crossover**). The selection of operators is also based on a roulette wheel with slots sized proportional to the probabilities of the operators. Note that in this version the time period for a single generation is the shortest.

G3.6.3 Development and implementation

The development of the EP/N is an ever-living 'evolution' process itself: different ideas have been experimented with and many improvements have been made since the earliest version, but there are still many new ideas and features that can be incorporated in the EP/N system (see section G3.6.5). Instead of seeking a complete product, we see the EP/N more as representing a new direction, along which there are many new hopes but also new challenges, and a new framework, under which these new hopes, in terms of new ideas and strategies, can be explored and tested, and the new challenges can be dealt with. Indeed, we have already discovered a mixture of hopes and challenges so far.

G3.6.3.1 Development of fitness function

The earliest version of the EP/N (Lin 1993, Lin *et al* 1994a, 1994b) can be characterized as having a single fitness criterion and a simple *penalty function*. Only the shortest distance criterion was used: the path cost was simply the length of the path, and a path was better than another one if it was shorter. (This was just as in many traditional approaches to path planning.) Infeasible paths were penalized by adding large penalty constants to their costs, making their lengths exceedingly long. C5.2

Such treatment hampered the ability of the EP/N to work well in difficult environments because of its many drawbacks. First, the shortest path may not be safe, that is, sufficiently away from obstacles, and it may not be more efficient than a longer path if it is not smooth. For example, in figure G3.6.6, the path q is longer than p but is obviously better. Hence, we changed the evaluation (or fitness) function to include factors of clearance and smoothness. We experimented with various ways of defining the fitness function and encountered the problem of how to evaluate the fitness of an infeasible path, for which clearance and smoothness do not make much sense. This investigation deepened our understanding of the problems introduced by using simple penalties to discriminate against infeasible paths.

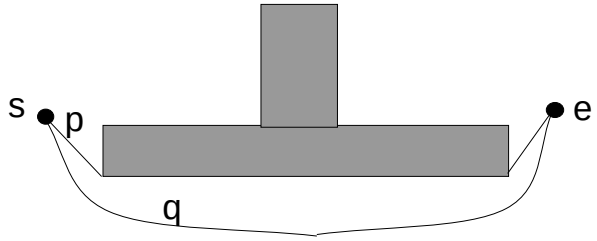


Figure G3.6.6. The longer path is better.

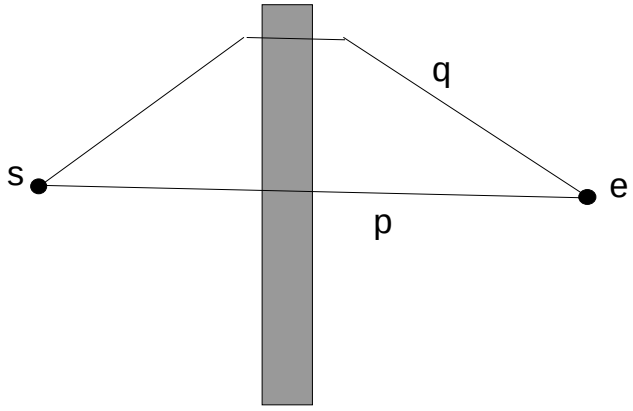


Figure G3.6.7. Two infeasible paths.

The major problem with using a simple penalty function is that it does not provide a reasonable basis for comparing two infeasible paths, since the merit of a path is not merely reflected by its length and what makes one feasible path better than another simply may not be applicable to the comparison of two infeasible paths. (In fact, as we do not even consider comparing two infeasible paths in our daily lives, we have much less intuition to help us than in the case of comparing two feasible ones.) For example, in figure G3.6.7, path p has the shortest distance (a straight line) and a perfect smoothness, if smoothness is counted. The other path q has longer distance and worse smoothness. Thus, with the same constant penalty on both paths, p will be ranked better than q, although it seems that q is actually better in the sense that q can be mutated into a feasible path relatively easily.

One may ask what will happen if we simply eliminate infeasible paths altogether and only evolve the feasible ones. Unfortunately, in our problem, except for cases with very simple environments which have only a few obstacles, the randomly generated initial population usually consists of infeasible paths only, and since the feasible solution space is nonconvex and has a complex boundary depending on obstacles, it is often more difficult to produce/reproduce only feasible paths than to deal with infeasible ones. Therefore, evaluating infeasible paths is extremely important and almost inevitable. Another important incentive for evolving infeasible paths is that it can speed up the search for the optimum solution by providing 'shortcuts' across the infeasible solution space.

Hence, our investigation results in the current solution of evolving feasible and infeasible paths by separate fitness functions as described in section G3.6.2.2.

G3.6.3.2 Development of operators

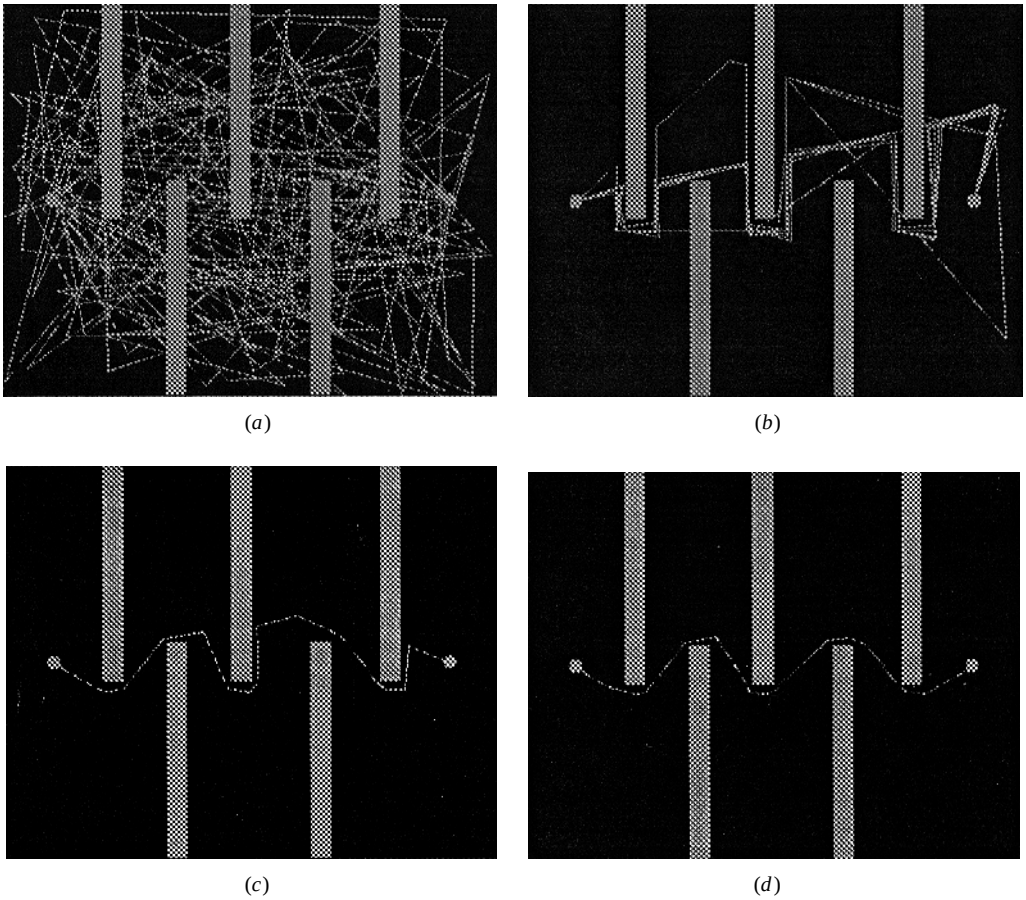
Initially, the first six operators were used in the EP/N. Different schemes and probabilities of applying those operators were experimented, and the effects of the operators were investigated. We later added the **smooth** operator to improve feasible paths. We tested the operators in different environments and found that for complex planning tasks in certain complex environments purely random operators did not work very well. This led us to design operators using more knowledge about the environment. The **repair** operator was introduced using knowledge of obstacles. In fact, we found that much of the knowledge

needed by more 'intelligent' operators had already been made available during evaluation of path fitness. In the latest version of EP/N, we added new versions for **mutation_1** (as explained in section G3.6.2.2 about genetic operators), as well as for **deletion** and **smooth** using such knowledge.

Our experience showed that **repair** was highly effective in generating feasible paths; **smooth** and the more 'intelligent' version of **mutation_1** were highly effective in improving feasible paths; **crossover** was consistently effective in evolving both infeasible and feasible paths. This was particularly important since **crossover** was completely random. Its simplicity also seemed to speed up considerably the evolution process. The only operator that removed nodes from a path was **deletion**, which thus was highly effective in keeping the EP/N system efficient (in time and space) and allowing other operators to be active. It seemed that the combined effort of different operators generally worked better in complex situations. As mentioned in section G3.6.2.2, how to determine the best combination of operators (i.e. probabilities) is not a trivial issue and is definitely one of the major future research topics (see section G3.6.5).

G3.6.3.3 Implementation

The earlier versions of the EP/N program were run on 486 or Pentium PCs. The later versions of the EP/N were run under Unix on Sun SparcStations. No commercial EA tools were used.



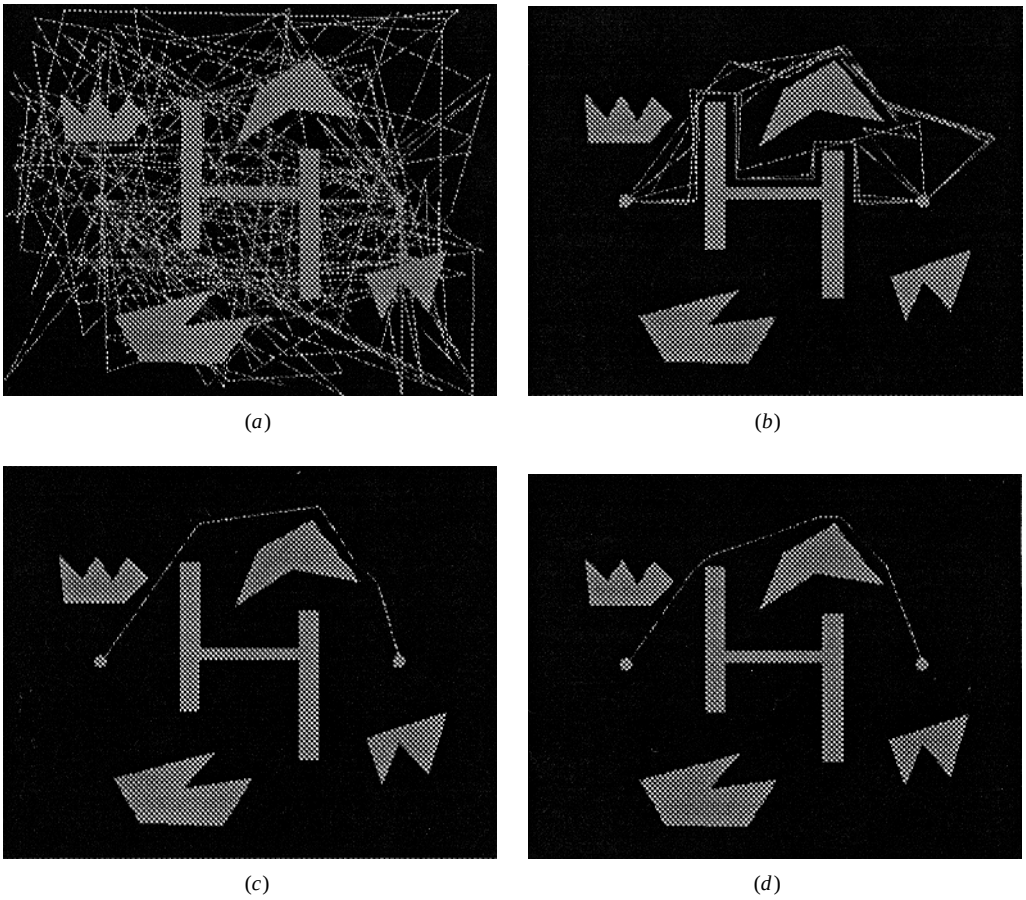
(*a*) (*b*)

(*c*) (*d*)

Figure G3.6.8. (*a*) $T = 0$: paths are generated randomly. (*b*) $T = 100$: evolution has taken 0.91 seconds. (*c*) $T = 600$: evolution has taken 14.67 seconds; the best path has 25 nodes and a cost of 630.19. (*d*) $T = 1000$: evolution has taken 28.16 seconds; the best path has 20 nodes and a cost of 598.62.

G3.6.4 Results

In figures G3.6.8–G3.6.11, we present some off-line planning results obtained from running the latest version of the EP/N system on a Sun Sparc 20 in different environments with the same set of parameter values as follows:



(*a*) (*b*)

(*c*) (*d*)

Figure G3.6.9. (*a*) $T = 0$: paths are generated randomly. (*b*) $T = 150$: evolution has taken 2.13 seconds. (*c*) $T = 300$: evolution has taken 3.92 seconds; the best path has four nodes and a cost of 483.95. (*d*) $T = 500$: evolution has taken 7.06 seconds; the best path has five nodes and a cost of 473.88.

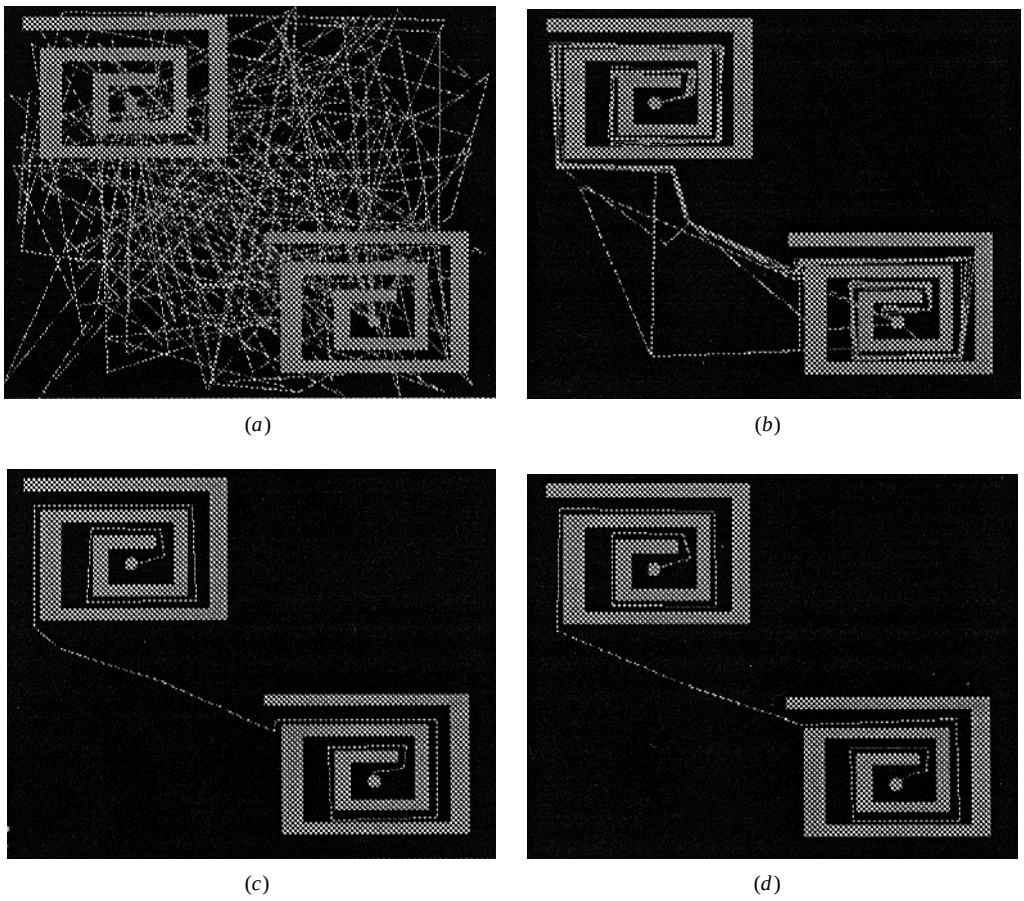
- probabilities of application for operators **crossover**, **mutation_1**, **mutation_2**, **insertion**, **deletion**, **swap**, **smooth**, and **repair** are 0.6, 0.8, 0.5, 0.5, 0.5, 0.5, 0.9, and 0.8 respectively
- population size is 30
- coefficients $w_{\mathrm{d}}, w_{\mathrm{s}}, w_{\mathrm{c}}, a$, and τ in the evaluation function $\mathrm{eval}_{\mathrm{f}}(p)$ are 1.0, 1.0, 1.0, 7.0, and 10 respectively.

Snapshots were taken at four different states, indicated by four different values of generation index T, of evolution for each task/environment, where two-thirds of the population were displayed at states (*a*) and (*b*), and only the best path was displayed at states (*c*) and (*d*). Despite the fact that the parameter values were chosen rather arbitrarily and the same 'one size fit all' values were applied to different environments with no individual adjustment, the EP/N system performed quite well as clearly shown by the results. Especially noteworthy is the efficiency the EP/N demonstrated in finding feasible paths as shown in states (*b*) and the near-optimal paths as shown in states (*c*). From states (*c*) to states (*d*), however, the pace of evolution was much slowed as expected.

G3.6.5 Conclusions

The EP/N represents a promising new approach in robot planning which is full of potential and a new application of evolutionary computation concepts which is full of interesting challenges. The EP/N is remarkably robust despite imperfections in the design of evaluation functions, the design and application (i.e. probabilities) of genetic operators, and the like. It confirms the nature and advantage of an evolutionary system.

One important issue in future research is how to further use domain knowledge (i.e. specific environmental knowledge) effectively in the EP/N system to improve performance. Although in our latest



(*a*) (*b*)

(*c*) (*d*)

Figure G3.6.10. (*a*) $T = 0$: paths are generated randomly. (*b*) $T = 100$: evolution has taken 1.60 seconds. (*c*) $T = 350$: evolution has taken 9.88 seconds; the best path has 19 nodes and a cost of 2434.88. (*d*) $T = 1000$: evolution has taken 27.45 seconds; the best path has 17 nodes and a cost of 2381.53.

version of the EP/N we incorporated domain knowledge in both fitness evaluation and genetic operators, there are other components/processes, such as initialization process and determination of parameter values, which may benefit from domain knowledge. For example, rather than random initialization, an initial population may consist of (i) a set of paths created by mutating or repairing the shortest path between start and goal locations or (ii) some mixture of chromosomes having randomly generated coordinates and chromosomes having coordinates with 'problem specific knowledge' as obtained from (i).

It is highly desirable to make the EP/N capable of adapting its parameter values based on domain knowledge and the states of evolution. Currently, all operators of the EP/N have constant probabilities of application, which are fixed at the beginning of an evolution process. However, different operators may have different impacts (roles) at different stages of the evolution process due to different situations encountered in an environment. For example, in on-line navigation, if the robot follows the current best path without running into any unexpected obstacles, the significance of **mutation_1** should grow, whereas the probability of **mutation_2** should be kept at the minimum level. On the other hand, if the robot is trapped in some location of the environment (e.g. surrounded by previously unknown obstacles), the probability of **mutation_2** should increase; at the same time the significance of **mutation_1** could shrink. While the role of **repair** should be very significant at the early stage of evolution, the role of **smooth** should become more significant at the later stage when the population consists of more feasible chromosomes. Similar observations and comments could be made for the other parameters.

Another important issue is to improve the organization of the EP/N to stress adaptability and learning for on-line navigation. For example, instead of generating a subpath for the robot to 'get around' an obstacle as is the case in the current version, the NEG may simply generate an alternative path for the robot to reach its goal, where the path is based on the 'past experience', which can be a pool of feasible paths obtained previously. It could also be interesting to study other forms of 'memory', such as one based

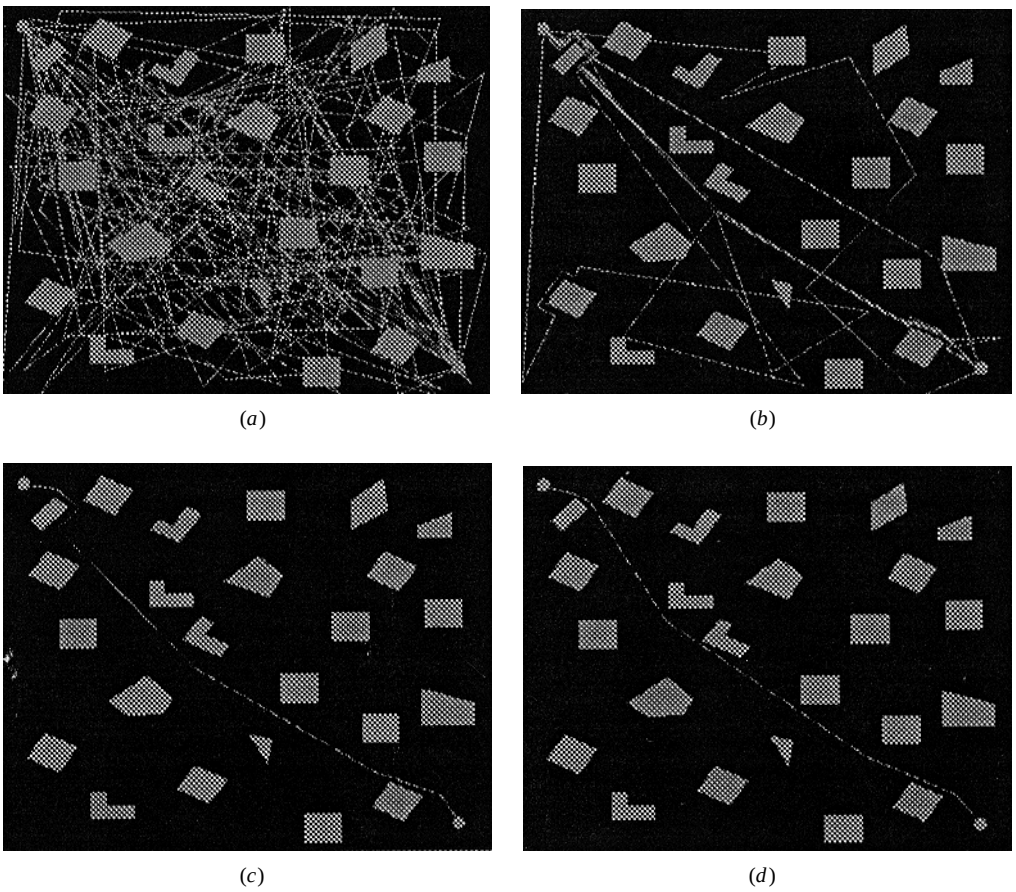


Figure G3.6.11. (*a*) $T = 0$: paths are generated randomly. (*b*) $T = 60$: evolution has taken 1.74 seconds. (*c*) $T = 150$: evolution has taken 6.19 seconds; the best path has seven nodes and a cost of 950.79. (*d*) $T = 400$: evolution has taken 19.78 seconds; the best path has six nodes and a cost of 910.60.

on multichromosome structures with a dominance function (Goldberg 1989) or one employing machine learning techniques.

Acknowledgement

The author would like to thank Zbigniew Michalewicz and Lixin Zhang for their important contribution to the improvement and implementation of the latest version of the EP/N.

References

Goldberg D E 1989 *Genetic Algorithms in Search, Optimization and Machine Learning* (Reading, MA: Addison Wesley)

Latombe J C 1991 *Robot Motion Planning* (Deventer: Kluwer)

Lin H-S 1993 *Dynamic Path Planning for a Mobile Robot Using Evolution Programming* Master Thesis, UNCC

Lin H-S, Xiao J and Michalewicz Z 1994a Evolutionary navigator for a mobile robot *Proc. IEEE Int. Conf. Robotics and Automation (San Diego, 1994)* (Piscataway, NJ: IEEE) pp 2199–204

——1994b Evolutionary algorithm for path planning in mobile robot robot environment *Proc. 1st IEEE Conf. on Evolutionary Computation (part of the IEEE World Congress on Computational Intelligence) (Orlando, FL, 1994)* (Piscataway, NJ: IEEE) pp 211–6

Michalewicz Z 1994 *Genetic Algorithms + Data Structures = Evolution Programs* 2nd edn (Berlin: Springer)

Pavlidid T 1982 *Algorithms for Graphics and Image Processing* (New York: Computer Science)

Yap C-K 1987 Algorithmic motion planning *Advances in Robotics, vol 1: Algorithmic and Geometric Aspects of Robotics* ed J T Schwartz and C-K Yap (Hillsdale, NJ: Erlbaum) pp 95–143

G3.7 Evolutionary robotics

Philip Husbands, Inman Harvey, Nicholas Jakobi, Adrian Thompson and Dave Cliff

Abstract

This case study introduces the field of evolutionary robotics. A specialized piece of robotic equipment for evolving visually guided behaviors is described. The results of successful experiments in the incremental evolution of target tracking and distinguishing behaviors are presented.

G3.7.1 Project overview

G3.7.1.1 Evolutionary robotics

This section introduces the field of evolutionary robotics through a case study. The focus will be on a particular example of evolving visually guided behaviors in an autonomous robot. The basic notion of evolutionary robotics is captured in figure G3.7.1. The evolutionary process, based on a *genetic algorithm* B1.2 (Holland 1975), involves evaluating, over many generations, whole populations of control systems specified by artificial genotypes. These are interbred using a Darwinian scheme in which the fittest individuals are most likely to produce offspring. Fitness is measured in terms of how good a robot's behavior is according to some evaluation criterion. The work reported here forms part of a long-term study to explore the viability of such an approach in developing interesting adaptive behaviors in visually guided autonomous robots, and, through analysis, in better understanding general mechanisms underlying the generation of such behaviors. It is one of the strands of the research program of the Evolutionary and Adaptive Systems Group, School of Cognitive and Computing Sciences, University of Sussex.

The motivations underlying our work have been discussed at length in a number of previous papers (e.g. Cliff *et al* 1993, Husbands *et al* 1995). Briefly, the argument goes like this. Traditional approaches to the development of control systems for autonomous mobile robots have made only modest progress, with fragile and computationally very expensive methods. This is due largely to the implicit assumption of functional decomposition—the assumption that perception, planning, and action can be analyzed and synthesized independently of each other. We strongly suspect, along with most people working in the areas of adaptive behavior and artificial life (Meyer and Wilson 1991, Meyer *et al* 1993, Cliff *et al* 1994, Langton 1989, Langton *et al* 1991, Langton 1994, Brooks and Maes 1994, Varela and Bourgine 1992, Moran *et al* 1995), many biologists (Young 1989, Ewert 1980), and increasing numbers of cognitive scientists (Sloman 1992), that useful control systems to generate sophisticated behaviors in such robots will necessarily involve many *emergent* interactions between many constituent parts (even though there may be hierarchical functional decomposition within some of these parts). However, we go further by claiming that there is no evidence that humans are capable of designing systems with these characteristics using traditional analytical approaches: hence the attraction of artificial evolution as an automatic alternative to hand design. There is no need for any assumptions about means to achieve a particular kind of behavior, as long as this behavior is directly or implicitly included in the evaluation function.

There are many different ways of realizing each stage of the cycle shown in figure G3.7.1. A crucial decision is whether or not to use simulation at the evaluation stage, transferring the end

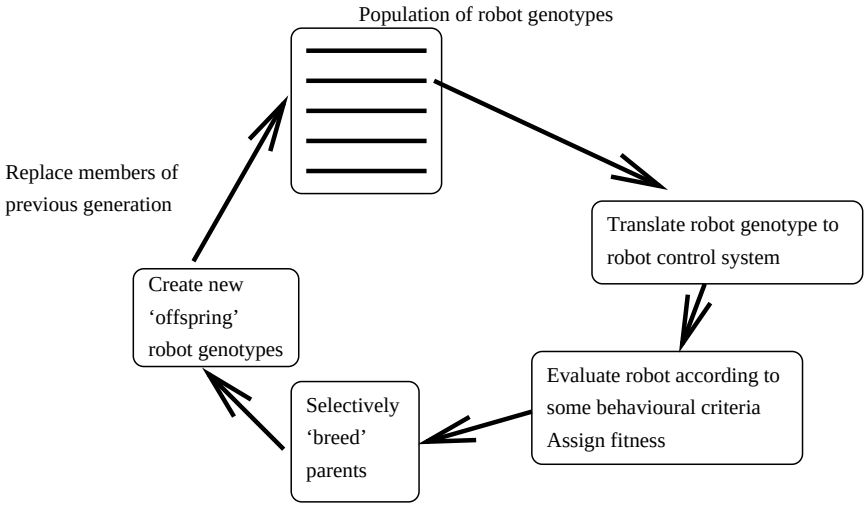


Figure G3.7.1. The basic notion of evolutionary robotics.

results to the real world. Since an evolutionary approach potentially requires the evaluation of populations of robots over many generations, a natural first thought is that simulations will speed up the process, making it more feasible. Despite initial scepticism (Brooks 1992), it has recently been shown that control systems evolved in carefully constructed simulations, with an appropriate treatment of noise, transfer extremely well to reality, generating almost identical behaviors in the real robot (Jakobi *et al* 1995, Thompson 1995). However, both of these examples involved relatively simple robot–environment interaction dynamics. Once even low-bandwith vision is used, simulations become altogether more problematic. They become difficult and time consuming to construct and computationally very intensive to run. Hence evolving visually guided robots in the real world becomes a more attractive option. This case study revolves around a piece of robotic equipment specially designed to allow the real-world evolution of visually guided behaviors—the Sussex gantry robot.

G3.7.1.2 The species adaption genetic algorithm: an incremental approach

Just as natural evolution involves adaptations to existing species, we believe genetic algorithms (GAs) (or some other form of evolutionary algorithm (EA)) should be used as a method for searching the space of possible adaptations of an existing *robot*, not as a search through the complete space of *robots*: successive adaptations over a long timescale can lead to long-term increases in complexity. For this reason, whereas most GAs operate on fixed-length genotypes, we believe it is necessary to work instead with variable-length genotypes. This leads to an incremental approach. A series of gradually more demanding task-based evaluation schemes are used. In this way new capabilities are built on existing ones and the search space is always constrained enough to be manageable.

The basis for extending standard GAs to cope with this has been worked out by Harvey (1992b, 1994, 1992a), which describe the *species adaptation genetic algorithm* (SAGA). In SAGA, the population being evolved is always a fairly genetically converged *species*; and increases in genotype length (or other metric of expressive power), associated with increases in complexity, can occur only very gradually.

G3.7.1.3 What to evolve: artificial neural networks

When relying on evolution for the design of a control system, appropriate building blocks must be chosen for it to work with. Some have advocated *classifier systems* (Dorigo and Schnepf 1993; see Goldberg 1989 for a coverage of classifier systems). Some propose LISP-like *programming languages* (Koza 1992, Brooks 1992). Beer and Gallagher (1992) and Yamauchi and Beer (1994) have used *dynamical neural networks*. B1.5.2 B1.5.1 D1

Details of our reasoning in deciding on control system primitives can be found in the articles by Cliff *et al* (1993), and Husbands *et al* (1995). Briefly, our criteria are

- they are the primitives of a dynamical system
- the system should operate in real time
- the system should be 'evolvable', not 'brittle'; in the sense that many of the possible small changes in the way components are bolted together should result in only small changes in resulting behavior
- incremental change in the complexity of any structure composed of such primitives should be possible.

There may be many possible components and general architectures that meet these criteria. The particular choice we have focused on is that of recurrent dynamic real-time networks, where the primitives are the nodes in a network, and links between them. There are no restrictions on network topologies, arbitrarily recurrent nets being allowed. When some of these nodes are connected to sensors, and some to actuators, the network acts as a control system, generating behaviors in the robot.

G3.7.2 Design process

G3.7.2.1 Concurrent evolution of visual morphologies and control networks

Rather than imposing a fixed visual sampling morphology (geometric layout of the visual sensors), we believe a more powerful approach is to allow the visual morphology to evolve along with the rest of the control system. Hence we genetically specify regions of the robot's visual field to be subsampled, these provide the only visual inputs to the control network. It would be desirable to have many aspects of the robot's morphology under genetic control, although this is not yet technically feasible.

G3.7.2.2 The gantry robot

The gantry robot is shown in figure G3.7.2. The robot is cylindrical, some 150 mm in diameter. It is suspended from the gantry frame with stepper motors that allow translational movement in the X and Y directions, relative to a coordinate frame fixed to the gantry. The maximum X (and Y) speed is about 200 mm s^{-1}. Such movements, together with appropriate rotation of the sensory apparatus, correspond to those which would be produced by left and right wheels. The visual sensory apparatus consists of a charge-coupled device (CCD) camera pointing down at a mirror inclined at 45° to the vertical (see figure G3.7.3). The mirror can be rotated about a vertical axis so that its orientation always corresponds to the direction the 'robot' is facing. The visual inputs undergo some transformations *en route* to the control system, described later. The hardware is designed so that these transformations are performed completely externally to the processing of the control system.

Figure G3.7.2. A view of the gantry. The horizontal girder moves along the side rails, and the robot is suspended from a platform which moves along this girder.

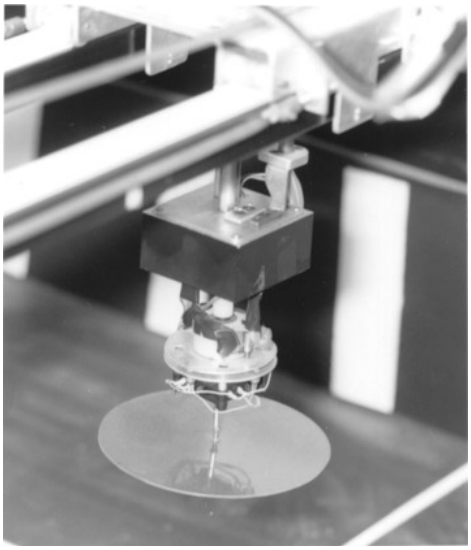


Figure G3.7.3. The gantry robot. The camera inside the top box points down at the inclined mirror, which can be turned by the stepper motor beneath. The lower plastic disk is suspended from a joystick, to detect collisions with obstacles.

The control system for the robot is run off-board on a fast personal computer, the *brain PC*. This computer receives any changes in visual input by interrupts from a second dedicated *vision PC*. A third (single-board) computer, the SBC, sends interrupts to the brain PC signalling tactile inputs resulting from the robot bumping into walls or physical obstacles. The only outputs of the control system are motor signals. These values are sent, via interrupts, to the SBC, which generates the appropriate stepper motor movements on the gantry.

The roles of the three computers are illustrated in figure G3.7.4. Continuous visual data are derived from the output of the small monochrome CCD camera. A purpose-built frame grabber transfers a 64×64 image at 50 Hz into a high-speed 2 kbyte complementary metal oxide semiconductor (CMOS) dual-port random access memory (RAM), completely independently and asynchronously relative to any processing of the image by the vision PC. The brain PC runs the top-level genetic algorithm and during an individual evaluation it is dedicated to running a genetically specified control system for a fixed period. At intervals during an evaluation, a signal is sent from the brain PC to the SBC requesting the current position and orientation of the robot. These are used in keeping score according to the current fitness function. The brain PC receives signals, to be fed into the control system, representing sensory inputs from the vision PC and the SBC. The visual signals are derived from averaging over genetically specified circular receptive patches in the camera's field of view.

This setup, with off-board computing and avoidance of tangled umbilicals, means that the apparatus can be run continuously for long periods of time—making artificial evolution feasible. A top-level program automatically evaluates, in turn, each member of a population of control systems. A new population is produced by selective interbreeding and the cycle repeats. For full technical details of the system see the article by Harvey *et al* (1994).

G3.7.2.3 The artificial neural networks

The artificial neurons used have separate channels for excitation and inhibition. Real values in the range $[0, 1]$ propagate along excitatory links subject to delays associated with the links. The inhibitory (or veto) channel mechanism works as follows. If the sum of excitatory inputs exceeds a threshold, T_v, the value 1.0 is propagated along any inhibitory output links the unit may have; otherwise a value of 0.0 is propagated. Veto links also have associated delays. Any unit that receives a nonzero inhibitory input has its excitatory output reduced to zero (i.e. is vetoed). In the absence of inhibitory input, excitatory outputs are produced by summing all excitatory inputs, adding a quantity of noise, and passing the resulting sum through a simple linear threshold function, $F(x)$, given below. Noise was added to provide further potentially interesting

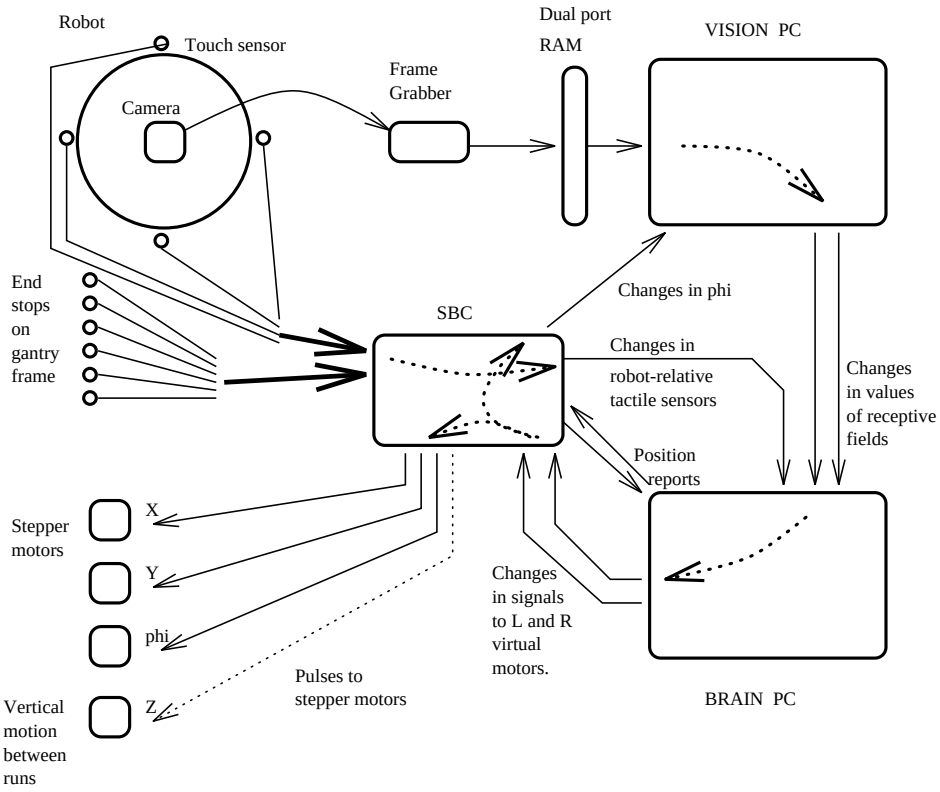


Figure G3.7.4. The different roles of the vision computer, the brain computer, and the SBC.

and useful dynamics. The noise was uniformly distributed in the real range $[-N, +N]$.

$$F(x) = \begin{cases} 0 & \text{if } x \leq T_1 \\ \dfrac{x - T_1}{T_2 - T_1} & \text{if } T_1 < x < T_2 \\ 1 & \text{if } x \geq T_2. \end{cases} \tag{G3.7.1}$$

The networks' continuous nature was modeled by using very fine-time-slice techniques. In the experiments described in this paper the following neuron parameter settings were used: $N = 0.1$, $T_v = 0.75$, $T_1 = 0.0$ and $T_2 = 2.0$. The networks are hard-wired in the sense that they do not undergo any architectural changes during their lifetime; they all had unit weights and time delays on their connections. These networks are just one of the class, outlined in section G3.7.1.3, that we are interested in investigating.

G3.7.2.4 The genetic encoding

Two *chromosomes* per robot are used. One of these is a fixed-length bitstring encoding the position and size of three visual receptive patches as described above. Three eight-bit fields per patch are used to encode their radii and polar coordinates in the camera's circular field of view. The other chromosome is a variable-length character string encoding the network topology. The genetic encoding used for the control network is illustrated in figure G3.7.5.

The network chromosome is interpreted sequentially. First the input units are coded for, each preceded by a marker. For each node, the first part of its gene can encode node properties such as threshold values; there then follows a variable number of character groups each representing a connection from that node. Each group specifies whether it is an excitatory or veto connection, and then the target node indicated by jump type and jump size. In a manner similar to that used by Harp and Samad (1992), the jump type allows for both relative and absolute addressing. Relative addressing is provided by jumps forwards or backwards along the genotype order; absolute addressing is relative to the start or end of the genotype. These modes of addressing mean that offspring produced by crossover will always be legal. There is one input node for each sensor (three visual; four tactile).

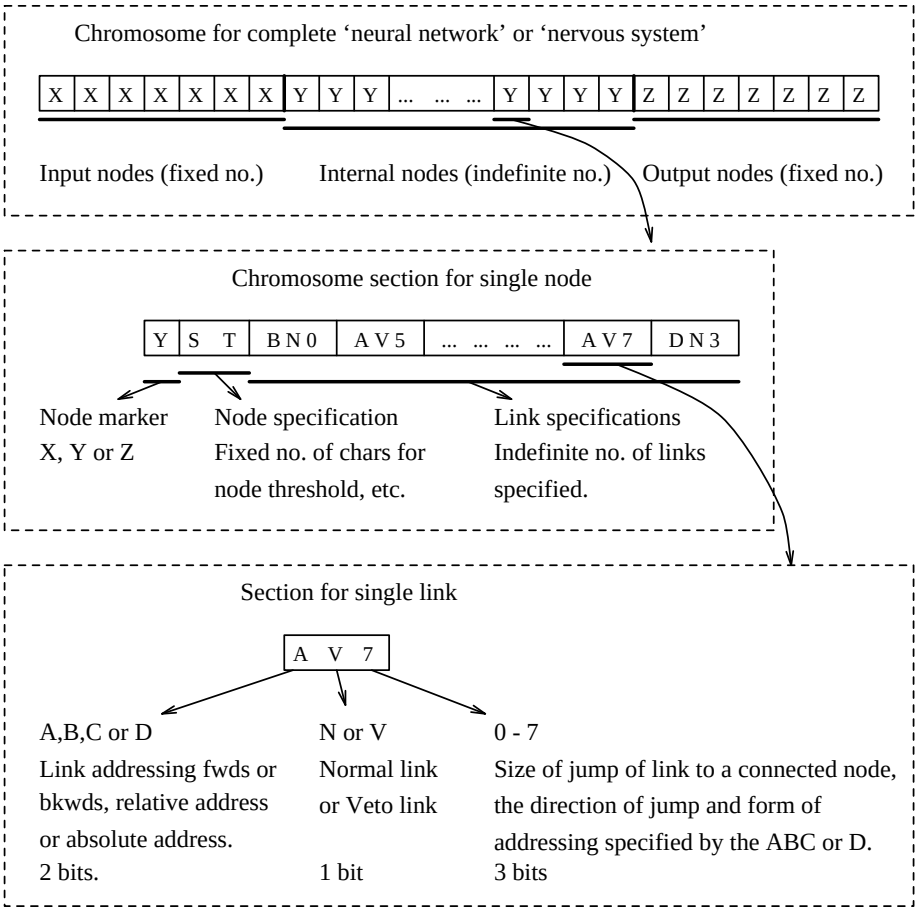


Figure G3.7.5. The genetic encoding scheme.

The internal nodes and output nodes are handled similarly with their own identifying genetic markers. Clearly this scheme allows for any number of internal nodes. The variable length of the resulting genotypes necessitates a careful crossover operator which exchanges homologous segments. In keeping with SAGA principles, when a crossover between two parents can result in an offspring of different length, such changes in length (although allowed) are restricted to a minimum (Harvey 1992a). There are four output neurons, two per motor. The outputs of each pair are differenced to give a signal in the range $[-1, 1]$.

G3.7.2.5 Experimental setup

In each of the experiments a population size of 30 was used with a GA employing a linear rank-based selection method, ensuring the best individual in a population was twice as likely to breed as the median individual. Each generation took about 1.5 h to evaluate. The most fit individual was always carried over to the next generation unchanged. A specialized crossover allowing small changes in length between offspring and parents was used (Cliff *et al* 1993). Mutation rates were set at 1.0 bit per vision chromosome and 1.8 bits per network chromosome.

With the walls and floor of the gantry environment predominantly dark, initial tasks were navigating towards white paper targets. In keeping with the incremental evolutionary methodology, deliberately simple visual environments are used initially, as a basis for moving on to more complex ones. Illumination was provided by fluorescent lights in the ceiling above, with the gantry screened from significant daylight variations. However, the dark surfaces did not in practice provide uniform light intensities, either over space or over time. Even when the robot was stationary, individual pixel values would fluctuate by up to 13%.

G3.7.3 Results

G3.7.3.1 A large target

In the first experiment, one long gantry wall was covered with white paper. The evaluation function $\mathcal{E}_1$, to be maximized, implicitly defines a target locating task, which we hoped would be achieved by visuomotor coordination:

$$\mathcal{E}_1 = \sum_{i=1}^{20} Y_i \tag{G3.7.2}$$

where Y_i are the perpendicular distances of the robot from the wall opposite that to which the target is attached, sampled at 20 fixed time intervals throughout a robot trial which lasted a total of about 25 s. The closer to the target the higher the score. For each robot architecture four trials were run, each starting in the same distant corner, but facing in four different partially random directions, to give a range of starts facing into obstacle walls as well as towards the target. As the final fitness of a robot control architecture was based on the *worst* of the four trials (to encourage robustness), and since in this case scores accumulated monotonically through a trial, this allowed later trials among the four to be prematurely terminated when they bettered previous trials. In addition, any control systems that had not produced any movement by 1/3 of the way into a trial were aborted and given zero score.

The run was started from a converged population made entirely of clones of a single randomly generated individual picked out by us as displaying vaguely interesting behavior (but by no means able to do anything remotely like locating and approaching the target). In two runs using this method very fit individuals appeared in fewer than ten generations. From a start close to a corner, they would turn, avoiding contact with the walls by vision alone, then move straight towards the target, stopping when they reached it.

G3.7.3.2 A small target

The experiment continued from the stage already reached, but now using a much narrower target placed about 2/3 of the way along the same wall the large target had been on, and away from the robot's starting corner (see figure G3.7.6), with evaluation $\mathcal{E}_2$:

$$\mathcal{E}_2 = \sum_{i=1}^{20} (-d_i) \tag{G3.7.3}$$

where d_i is the distance of the robot from the centre of the target at one of the sampled instances during an evaluation run. Again, the fitness of an individual was set to the worst evaluation score from four runs with starting conditions as in the first experiment. The initial population used was the 12th generation from a run of the first experiment (i.e. we incrementally evolved on top of the existing behaviors).

Within six generations a network architecture and visual morphology had evolved displaying the behavior shown in figure G3.7.6. This control system was tested from widely varying random starting

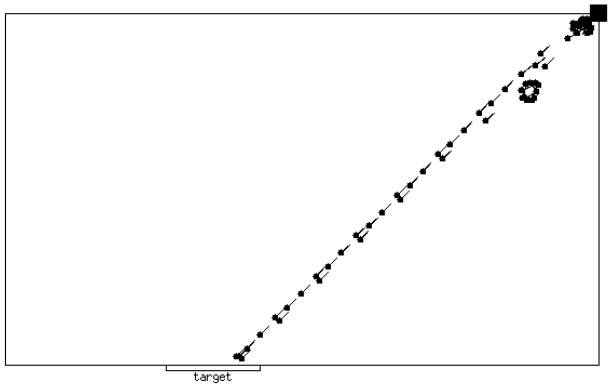


Figure G3.7.6. The behavior of the best of a later generation evolved under the second evaluation function. The dots, and trailing lines, show the front of the robot, and its orientation. Coarsely sampled positions from each of four runs are shown, starting in different orientations from the top right corner.

positions and orientations, with the target in different places, and with smaller and different shaped targets. Its behavior was general enough to cope with all these conditions for which it had not explicitly been evolved. It was also able to cope well with moving targets as shown in figures G3.7.7 and G3.7.8.

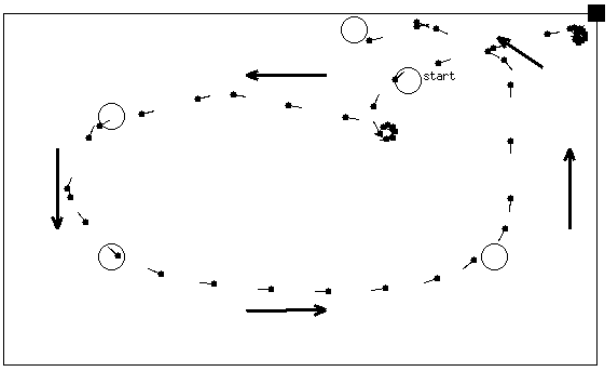


Figure G3.7.7. The tracking behavior of the control system that generated the behavior shown in the previous figure. The unfilled circles show the position of the target at a number of points on its path (the starting position is indicated). The arrows roughly indicate the path of the target.

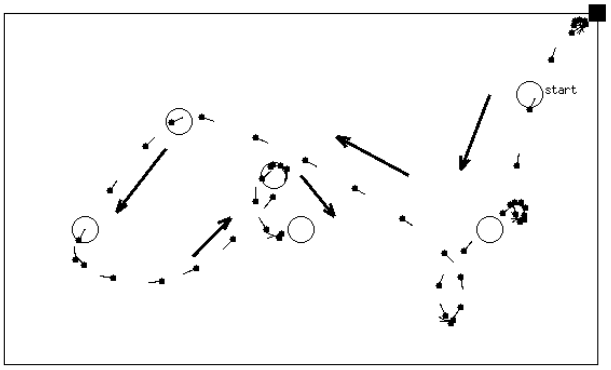


Figure G3.7.8. Further tracking behavior of the control system that generated the behavior shown in previous figure.

G3.7.3.3 Rectangles and triangles

The experiment continued with a distinguish-between-two-targets task. Two white paper targets were fixed to one of the gantry walls; one was a rectangle, the other was an isosceles triangle with the same base width and height as the rectangle. The robot was started at four positions and orientations near the opposite wall such that it was not biased towards either of the two targets. The evaluation function $\mathcal{E}_3$, to be maximized, was

$$\mathcal{E}_3 = \sum_{i=1}^{20} [\beta(D_{1_i} - d_{1_i}) - \sigma(D_{2_i},\, d_{2_i})] \tag{G3.7.4}$$

where D_1 is the distance of target 1 (in this case the triangle) from the gantry origin; d_1 is the distance of the robot from target 1; and D_2 and d_2 are the corresponding distances for target 2 (in this case the rectangle). These are sampled at regular intervals, as before. The value of function β is $(D_1 - d_1)$ unless d_1 is less than some threshold, in which case it is $3(D_1 - d_1)$. The value of σ (a *penalty function*) is zero unless d_2 is less than the same threshold, in which case it is $I - (D_2 - d_2)$, where I is the distance between the targets; I is more than double the threshold distance. High fitnesses are achieved for approaching the triangle but ignoring the rectangle. It was hoped that this experiment might demonstrate the efficacy of concurrently evolving the visual sampling morphology along with the control networks. C5.2

After about 15 generations of a run using as an initial population the last generation of the incremental small target experiment, fit individuals emerged capable of approaching the triangle, but not the rectangle,

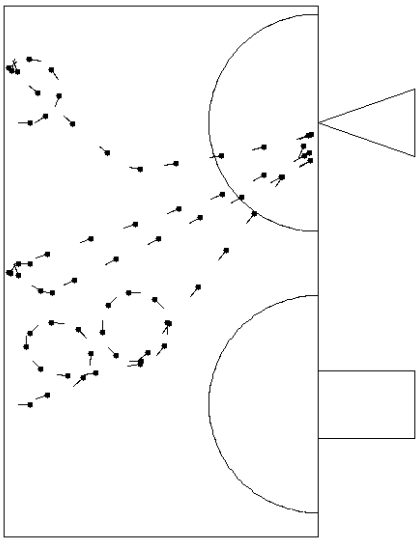


Figure G3.7.9. The behavior of a fit individual in the two-target environment. The rectangle and triangle indicate the positions of the targets. The semi circles mark the 'penalty' (near rectangle) and 'bonus score' (near triangle) zones associated with the fitness function. In these four runs the robot was started directly facing each of the two targets, and twice from a position midway between the two targets, once facing into the wall and once facing out.

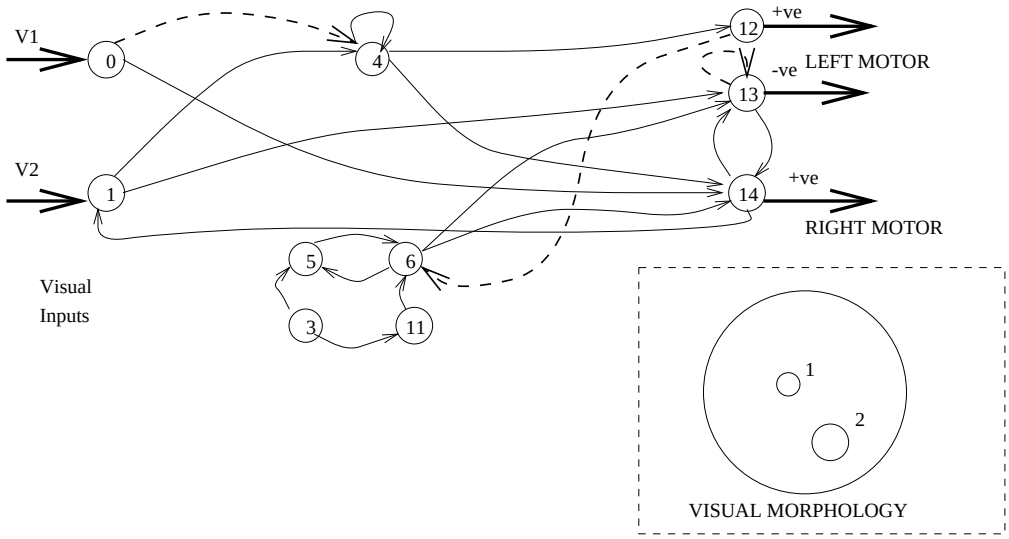


Figure G3.7.10. The active part of the control system that generated fit behavior for the rectangle and triangle experiment. The visual morphology is shown in the inset.

from each of the four widely spaced starting positions and orientations. The behavior generated by the fittest of these control systems is shown in figure G3.7.9. When started from many different positions and orientations near the far wall, and with the targets in different positions relative to each other, this controller repeatedly exhibited behaviors very similar to those shown.

The active part of the evolved network that generated this behavior is shown in figure G3.7.10. The evolved visual morphology for this control system is shown inset. Only receptive fields 1 and 2 were used by the controller.

Detailed analyses of this evolved system can be found in the articles by Harvey *et al* (1994) and Husbands (1996). To crudely summarize, unless there is a *difference* in the visual inputs for receptive fields 1 and 2, the robot makes rotational movements. When there is a difference it moves in a straight line. The visual sensor layout and network dynamics have evolved such that it fixates on the sloping edge of the triangle and moves towards it.

G3.7.4 Conclusions

This study has shown that simple robust visually guided behaviors can be evolved in the real world with surprisingly small populations and in very few generations. The evolved behaviors were all generated by extremely minimal vision systems and very small networks. We believe part of this encouraging success was due to a good choice of control system primitives. We believe a key element in future progress to more sophisticated behaviors will be more complex genotype to phenotype mappings (Husbands *et al* 1994, Gruau 1995, Kodjabachian and Meyer 1994).

Acknowledgement

This work was supported by EPSRC grant GR/J18125.

References

Beer R and Gallagher J 1992 Evolving dynamic neural networks for adaptive behavior *Adaptive Behavior* **2** 91–122

Brooks R A 1992 Artificial life and real robots *Proc. 1st Eur. Conf. on Artificial Life* ed F J Varela and P Bourgine (Cambridge, MA: MIT Press–Bradford) pp 3–10

Brooks R A and Maes P (eds) 1994 *Artificial Life IV* (Cambridge, MA: MIT Press–Bradford)

Cliff D, Harvey I and Husbands P 1993 Explorations in evolutionary robotics *Adaptive Behavior* **2** 73–110

Cliff D, Husbands P, Meyer J-A and Wilson S (eds) 1994 *From Animals to Animats 3: Proc. 3rd Int. Conf. on Simulation of Adaptive Behavior* (Cambridge, MA: MIT Press–Bradford)

Dorigo M and Schnepf U 1993 Genetic-based machine learning and behavior-based robotics: a new synthesis *IEEE Trans. Syst. Man Cybernet.* **SMC-23** 141–54

Ewert J-P 1980 *Neuroethology* (Berlin: Springer)

Goldberg D E 1989 *Genetic Algorithms in Search, Optimization and Machine Learning* (Reading, MA: Addison-Wesley)

Gruau F 1995 Automatic definition of modular neural networks *Adaptive Behavior* **3** 151–84

Harp S A and Samad T 1992 Genetic synthesis of neural network architecture *Handbook of Genetic Algorithms* ed L Davis (New York: Van Nostrand Reinhold) pp 202–21

Harvey I 1992a The SAGA cross: the mechanics of crossover for variable-length genetic algorithms *Parallel Problem Solving from Nature, 2 (Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature, Brussels, 1992)* ed R Männer and B Manderick (Amsterdam: Elsevier) pp 269–78

——1992b Species adaptation genetic algorithms: the basis for a continuing SAGA *Proc. 1st Eur. Conf. on Artificial Life* ed F J Varela and P Bourgine (Cambridge, MA: MIT Press–Bradford) pp 346–54

——1994 Evolutionary robotics and SAGA: the case for hill crawling and tournament selection *Artificial Life III (Santa Fe Inst. Studies Sci. Complexity, Proc. Vol. XVI)* ed C Langton (Redwood City, CA: Addison-Wesley) pp 299–326

Harvey I, Husbands P and Cliff D 1994 Seeing the light: artificial evolution, real vision *From Animals to Animats 3, Proc. 3rd Int. Conf. on Simulation of Adaptive Behavior* ed D Cliff, P Husbands, J-A Meyer and S Wilson (Cambridge, MA: MIT Press–Bradford) pp 392–401

Holland J 1975 *Adaptation in Natural and Artificial Systems* (Ann Arbor, MI: University of Michigan Press)

Husbands P 1996 Rectangle, triangles, robots and transients *Proc. Int. Symp. on Artificial Life and Robotics* ed M Sugisaka (Beppu) pp 252–5

Husbands P, Harvey I and Cliff D 1995 Circle in the round: state space attractors for evolved sighted robots *Robot. Autonomous Syst.* **15** 83–106

Husbands P, Harvey I, Cliff D and Miller G 1994 The use of genetic algorithms for the development of sensorimotor control systems *Proc. From Perception to Action Conf.* ed P Gaussier and J-D Nicoud (Los Alamitos, CA: IEEE Computer Society) pp 110–21

Jakobi N, Husbands P and Harvey I 1995 Noise and the reality gap: the use of simulation in evolutionary robotics *Advances in Artificial Life: Proc. 3rd Eur. Conf. on Artificial Life (Lecture Notes in Artificial Intelligence 929)* ed F Moran, A Moreno, J J Merelo and P Chacon (Berlin: Springer) pp 704–20

Kodjabachian J and Meyer J-A 1994 Development, learning and evolution in animats *Proc. From Perception to Action Conf.* ed P Gaussier and J-D Nicoud (Los Alamitos, CA: IEEE Computer Society) pp 96–109

Koza J 1992 *Genetic Programming: on the Programming of Computers by means of Natural Selection* (Cambridge, MA: MIT Press–Bradford)

Langton C G (ed) 1989 *Artificial Life: Proc. Interdisciplinary Workshop on the Synthesis and Simulation of Living Systems (Santa Fe Inst. Studies Sci. Complexity Vol. VI)* (Redwood City, CA: Addison-Wesley)

——1994 *Artificial Life III (Santa Fe Inst. Studies Sci. Complexity Vol. XVI)* (Redwood City, CA: Addison-Wesley)

Langton C G, Farmer J D, Rasmussen S and Taylor C (eds) 1991 *Artificial Life II (Santa Fe Inst. Studies Sci. Complexity Vol. XI)* (Redwood City, CA: Addison-Wesley)

Meyer J-A, Roitblat H and Wilson S (eds) 1993 *From Animals to Animats 2: Proc. 2nd Int. Conf. on Simulation of Adaptive Behavior* (Cambridge, MA: MIT Press–Bradford)

Meyer J-A and Wilson S 1991 *From Animals to Animats: Proc. 1st Int. Conf. on Simulation of Adaptive Behavior* (Cambridge, MA: MIT Press–Bradford)

Moran F, Moreno A, Merelo J J and Chacon P 1995 *Advances in Artificial Life: Proc. 3rd Eur. Conf. on Artificial Life (Lecture Notes in Artificial Intelligence 929)* (Berlin: Springer)

Sloman A 1992 *Silicon Souls: How to Design a Functioning Mind* CSRP-92-11 School of Compter Science, University of Birmingham

Thompson A 1995 Evolving electronic robot controllers that exploit hardware resources *Advances in Artificial Life: Proc. 3rd Eur. Conf. on Artificial Life (Lecture Notes in Artificial Intelligence 929)* ed F Moran, A Moreno, J J Merelo and P Chacon (Berlin: Springer) pp 640–56

Varela F and Bourgine P (eds) 1992 *Proc. 1st Eur. Conf. on Artificial Life* ed F J Varela and P Bourgine (Cambridge, MA: MIT Press–Bradford)

Yamauchi B and Beer R 1994 Sequential behavior and learning in evolved dynamical neural networks *Adaptive Behavior* **2** 219–46

Young D 1989 *Nerve Cells and Animal Behaviour* (Cambridge: Cambridge University Press)

G4.1 Tuning Monte Carlo generator parameters to measured data by using genetic algorithms

Siegfried Hahn

Abstract

Monte Carlo generators are important tools for analyzing the data measured in high-energy physics experiments. They describe complete physics events on the basis of various underlying physical models. All these event generators include several free parameters which cannot be predicted by theory but have to be determined by comparing simulated events with measured data. Adjusting these parameters is a difficult task due to the complicated nonlinear correlations between different parameters, the multimodal structure of the search space, and the statistical fluctuations of the quality function. The use of conventional fitting strategies for this optimization problem requires the knowledge, experience and to some extent the intuition of a human expert in order to reduce the huge amount of calculation time to a reasonable limit. In contrast, genetic algorithms offer an automated procedure which does not require any previous knowledge about the search space. The global character of the search procedure leads to several distinct solutions and reveals the multimodality of the parameter space. The specific sampling method of genetic algorithms allows a further analysis of the evaluated parameter sets, yielding additional information on correlations between parameters. This leads to a deeper insight into the physics context and allows the identification of the global optimum.

G4.1.1 Overview

In mid-1989 LEP, the largest electron–positron collider to date, was put into operation at CERN, the European Laboratory for Particle Physics near Geneva, Switzerland. In a subterranean tunnel with a circumference of 27 km, electrons and positrons—the latter consist of antimatter and are the counterparts of the electrons—are accelerated to almost the speed of light. Beams of electrons and positrons circulate in opposite orbits and are made to collide in the center of huge particle detectors. Every collision of these particles (a so-called event) releases energy up to an amount of 100 billion electronvolts. From an energy flash of that kind, several dozens of new elementary particles emerge which can be measured within those detectors.

The objective of the four LEP experiments, ALEPH, OPAL, DELPHI, and L3, is to evaluate the standard model of elementary particle physics, as well as to determine some important parameters of the theory with the utmost precision, such as the mass of the Z^0 boson, one of the fundamental interaction particles of the theory, which is generated in large numbers at the LEP collider. For a comprehensive overview of the LEP physics aims, see Altarelli *et al* (1989).

G4.1.2 Monte Carlo generators

Monte Carlo (MC) generators are indispensable tools for the comparison of the data arising in the course of the LEP experiments, as well as all other accelerator experiments in high-energy physics, with the theoretical predictions of the standard model. They are applied for the most diverse steps in the analysis

chain, such as for the classification and the selection of particular events or for studies concerning the influence of detector effects on the measurement.

MC generators are simulation programs, which describe the generation of complete physical events on the basis of the underlying physical models. The aim is to reflect as accurately as possible the experimental data in their wide variety and in the greatest detail, limited only by current knowledge of the underlying physical processes.

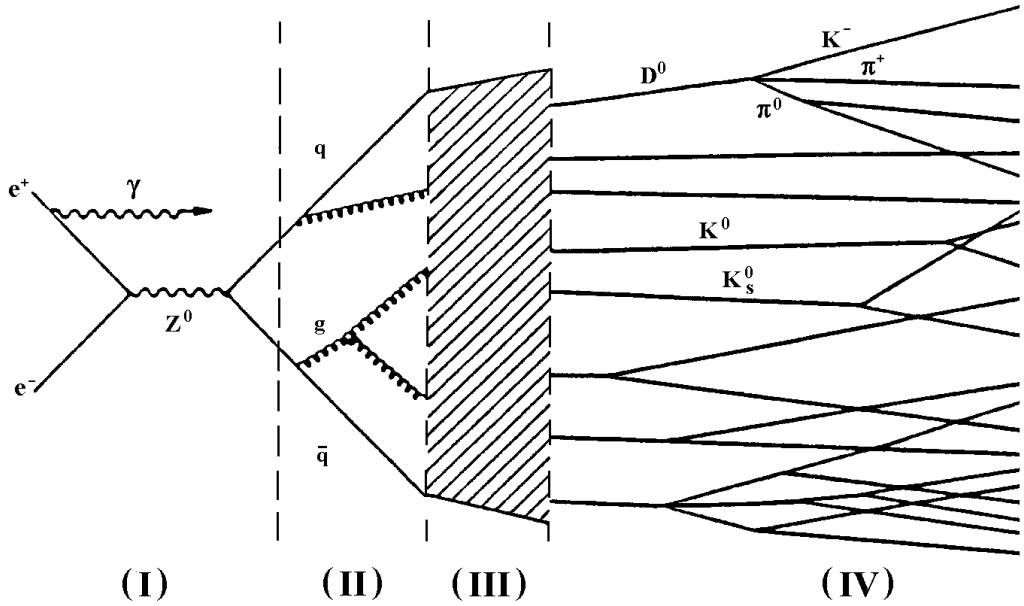


Figure G4.1.1. Schematic view at the four phases of particle production for the hadronic decay of the Z^0 in electron–positron annihilation: (I) Electroweak phase. (II) Perturbative QCD (quantum chromodynamics), which can be described by matrix element or partonshower models. (III) Nonperturbative QCD, which can be described by various heuristic fragmentation models (e.g. string fragmentation or cluster fragmentation). (IV) Decay of unstable particles, which can either be described through experimentally measured branching ratios or by various specialized heuristic decay models.

For the complete event generation, from the initial annihilation process to the final-state particles, the program is divided into several stages. For most of these stages there is a choice between different models corresponding to the different theoretical approaches for the processes under consideration. (See figure G4.1.1 for a schematic view of an example of the event generation process in electron–positron annihilation.) Every single reaction step proceeds nondeterministically; that is, the physical models determine only the probability of the various processes that might happen at every moment of the event generation. A random number generator decides which of the possible alternatives will be chosen at a given point. A comprehensive overview of MC generators can be found in Bambah *et al* (1989) or Sjöstrand (1989).

For our studies we used the JETSET 7.3 partonshower generator, which was developed at Lund University (see Sjöstrand 1992) and is one of the most frequently used MC generators for LEP physics.

G4.1.2.1 The tuning task

All current MC generators contain a number of free model parameters which cannot be predicted by theory. In order to determine their values, the simulated data have to be fitted to the measured distributions. If there is a good correspondence, the parameters may then be extracted from the generators.

For the measurement of the global event shape characteristics, as well as for single-particle properties, a large number of specialized observables have been developed. Due to the indeterminism of the basic processes, one cannot compare the MC generator output with the measured data on an event-by-event basis, but instead must compare distributions of these observables, generated from a set of several thousand events. The similarity between experimental and simulated distributions is measured by their χ^2 value

(per degree of freedom, d.f.) which is given by

$$\frac{\chi^2}{\text{d.f.}} = \frac{1}{N_d}\sum_{j=1}^{N_d}\left[\frac{1}{N_j}\sum_{i=1}^{N_j}\frac{\left(N_{\text{Data}}^{(i)} - N_{\text{MC}}^{(i)}\right)^2}{\left(\Delta N_{\text{Data}}^{(i)}\right)^2 + \left(\Delta N_{\text{MC}}^{(i)}\right)^2 + \left(\Delta N_{\text{Syst.}}^{(i)}\right)^2}\right] \tag{G4.1.1}$$

where N_d is the number of considered distributions, N_j is the number of individual bins of the jth histogram, $N^{(i)}$ is the content of the ith histogram bin either on MC or on data distributions, and $\Delta N^{(i)}$ is the statistical error (and the systematic error respectively) associated with the content of the ith histogram bin. A crucial point in tuning MC generator parameters to measured data is the choice of the set of distributions. In order to describe the data in full detail with high precision, one should choose a subset of observables so that approximately equal sensitivity to the parameters to be optimized is achieved. For our tuning we choose the following eight observables: thrust (T), oblateness (O), sphericity (S), aplanarity (A), rapidity (Y), the components of the transverse momentum in and out of the event plane ($p_\perp^{\text{in}}$, $p_\perp^{\text{out}}$) and the scaled momentum (x_p). For a precise definition of these entities, see DELPHI Collab. (1990).

G4.1.2.2 Statistical fluctuations

A major difficulty for traditional optimization techniques is the statistical fluctuation in the determination of the χ^2 value corresponding to the set of model parameters to be evaluated. Due to the indeterminism of the fundamental quantum mechanical processes, one would need to generate an infinite number of events to get exact χ^2 values. Therefore one must work principally with an approximate evaluation owing to the limited statistics. The dependency of the χ^2 value and its statistical error on the number of generated events is shown in figure G4.1.2.

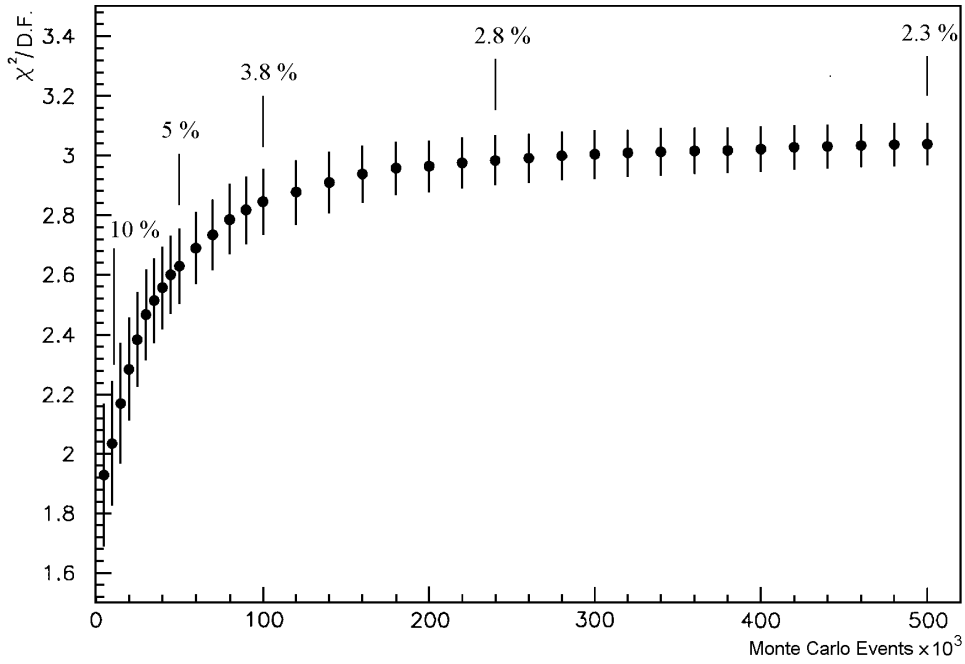


Figure G4.1.2. χ^2 value and statistical error as a function of the number of generated events.

The event generation process is a very time-consuming task. For a sample of 50 000 events, whose generation takes approximately one CPU hour on a DEC 3000 workstation, the relative statistical error of the χ^2 value is about 5%. The statistical error decreases only very slowly with the number of generated events, as seen in figure G4.1.2; the error is still more than 2%, even for a sample of 500 000 events. Here one has to consider that the optimization procedure requires the evaluation of χ^2 values for several hundred different sets of parameters, depending on the number of parameters to be tuned.

The robustness of *genetic algorithms* (GAs) with respect to statistical fluctuations plays an important B1.2 role in this optimization task. One major point is that GAs permit the evaluation of individuals with

different accuracy. That is because, for individuals with bad fitness values, it is not important for a GA to know exactly how bad they are, but it is only necessary to distinguish slight differences between those well-adapted individuals which determine the optimization process. This can be used in order to significantly reduce the computation time through a partial evaluation of the individuals; that is, the use of different levels of statistics depending on the individuals' quality. In this study we employed four different evaluation levels from 5000 up to 500 000 events per individual, which were applied according to individual fitness values as well as the average fitness of the populations.

One problem in this context is that the selection procedure of a GA is biased to favor individuals with overestimated fitness values. Since there is a high probability that just the best-performing individuals of a population are evaluated 'too good', this evaluation error would mislead the algorithm, if such individuals were to be propagated throughout the generations without an update of their fitness values. Therefore, every individual which has been taken into the new generation, without any modification, has to be evaluated again. The additional statistics can be used to update the individual's fitness value, so that potential evaluation errors decrease during the lifetime of an individual.

G4.1.3 Tuning strategies for Monte Carlo generators

Tuning seeks to determine of Monte Carlo model parameters that cannot be predicted by theory, in order to describe the experimental data as accurately as possible in their entire variety and in the greatest detail. This is done by a fit which minimizes the χ^2 value between generated and experimental distributions. The minimization is complicated by the fact that the models depend on probabilistic relations and thus have statistical fluctuations. Most of the model parameters are highly correlated and show multimodal behavior so that deterministic algorithms are likely to be misled.

G4.1.3.1 Conventional fitting strategies

Several attempts to tune MC generators have been made by using different strategies. A conventional fitting method is the simplex algorithm (e.g. as used in Fürstenau (1993)) from the program packet MINUIT (James 1994) which is known to be quite robust with respect to statistical fluctuations. Even so, it is not robust enough to tune more than one or two parameters simultaneously. Other methods use grids in the n-dimensional parameter space (see e.g. ALEPH Collab. 1992). Distributions are calculated for each point in the grid. The dependency for each bin of the distribution is then approximated by a second-order Taylor polynomial. Then a conventional algorithm is used to minimize the χ^2 of the parametrized bins. These methods have the disadvantage that the results depend strongly on the choice of the central point of the grid. This implies a human expert who already 'knows' good starting values for the parameters in advance.

Up to now, none of these algorithms has been able to tune both parameters a and b of the LUND fragmentation function, since they are highly correlated. For an exact definition of the LUND fragmentation function, see Sjöstrand (1992). Therefore, only one of these parameters had been tuned while the other was set to a randomly chosen value. In contrast, the results of our study (see Hahn 1993) indicate that it is necessary to tune both parameters to get the best possible correspondence between the MC model and the measured data.

G4.1.3.2 Genetic algorithms

Since there is an evident lack of suitable conventional algorithms for tuning MC generator parameters, GAs seem to be an interesting alternative. GAs maintain a whole population of search points by exploiting correlations between the individual members; therefore they are very efficient, particularly for problems with high dimensionality. Due to the global character of the search strategy, GAs handle complicated multimodal problems with no restrictions on the parameter's fit range. GAs do not require any problem-specific expert knowledge and can therefore be seen as a nearly objective optimization method. In contrast to conventional deterministic algorithms, the 'weakness' of the probabilistic sampling method causes tolerance towards sampling errors and therefore robustness in regard to statistical fluctuations.

In a first approach, we tuned the five most important QCD and fragmentation parameters of the JETSET 7.3 partonshower (PS) generator (Sjöstrand 1992) given in table G4.1.1. For a definition and description of their physical meaning, see, for example, DELPHI Collab. (1990), and for further details on the tuning prerequirements (preparation of experimental data, histogram binning, chosen options, etc.)

see Hahn (1993) and Hahn *et al* (1994). We performed a fit to the DELPHI data from 1992, using the eight distributions of global event shape and single-particle variables listed in subsection G4.1.2.1.

Table G4.1.1. Tuned parameters of the JETSET 7.3 partonshower (PS) generator, fit range and default values of the program.

Parameter	Default values	Fit range
Lund a	0.50	$0.1 - 0.8$
Lund b	0.90	$0.1 - 0.8$
σ_q	0.35	$0.1 - 0.5$
Q_0	1.00	$0.5 - 3.5$
Λ_{LLA}	0.40	$0.1 - 0.5$

Due to the statistical fluctuations in the quality function, only a limited accuracy can be achieved for the parameters to be tuned. Therefore, the number of bits allocated for the coding of each parameter in the individual's chromosome has to be chosen according to the desired resolution, thus preventing the algorithm from making too-small steps which would not change the related fitness values significantly.

First tests were performed using a C implementation of a simple GA as described in Goldberg (1989), using a *crossover rate* $p_c = 0.6$ and a mutation rate $p_m = 0.001$. The population size λ was varied between 50 and 80. The fitness function chosen was C3.3.1, C3.2.1

$$f(a_i) = \begin{cases} 1 - \chi^2(a_i)/\chi^2_{\max} & \text{if } \chi^2(a_i) \leq \chi^2_{\max} \\ 0 & \text{otherwise} \end{cases} \tag{G4.1.2}$$

resulting in a maximization task from the original minimization problem. Here $\chi^2(a_i)$ denotes the χ^2 value of individual a_i and $\chi^2_{\max}$ the maximum χ^2 value evaluated within the first generation. *Proportional selection* was chosen in combination with a linear fitness scaling. Furthermore, an elitist strategy was applied; that is, the guaranteed survival of the fittest individual. C2.2

For the tuning procedure it is not necessary to explicitly define termination criteria. By monitoring the search process one can terminate it either if the algorithm gets stuck or if a suitable solution is found.

The GA has been implemented by applying a parallel evaluation to the individuals on a DEC 3000 workstation cluster with nine nodes. Information exchange between the processes can easily be realized by the means of LAN communication, since most of the computation time is spent on function evaluation, for each individual independently of the other processes.

It turned out that the strategy described above was not capable of finding appropriate solutions to our optimization task. The algorithm lacked robustness with regard to statistical fluctuations; in particular, individuals which had been evaluated too good were misleading the algorithm, which then always resulted in premature convergence.

G4.1.3.3 Sharing models

One method to improve the quality of convergence in the presence of noise is the enlargement of the *population size* (Goldberg *et al* 1991). This was not considered here, since it was expected to impair the computational performance too much. E1.1

A quite natural way of maintaining the populations' diversity is the introduction of *fitness sharing*, in particular with respect to the multimodal structure of the search space. Here the fitness of each individual is scaled according to its mean distance to the whole population. In consequence, an increased competition takes place between individuals near by in the search space and less competition between individuals which are more 'unique' in comparison with the other members of the population. (For a detailed discussion of the sharing method, see, for example, Goldberg and Richardson 1987.) C6.1.2

Applying this method, several advantages are to be observed. It helps to maintain a more diverse population without increasing the population size or introducing more random elements into the search, such as a higher mutation rate would do. Since the attractive potential of a single point in the search space is reduced, the search is more tolerant towards individuals with overestimated fitness and therefore more robust with respect to statistical fluctuations. Perhaps the most important effect is that the global character of the search procedure is reinforced; that is, several individuals representing different (suboptimal) solutions

of the optimization task can be maintained within the population, which may recombine to build globally optimal solutions.

In combination with an increased crossover and mutation rate ($p_c = 0.95$, $p_m = 0.01$), the population size can even be reduced to $\lambda = 30$, resulting in an additional performance increase. For more details on the strategy parameters of the GA, see Hahn *et al* (1992).

G4.1.4 Results

In the course of 12 optimization runs with partially different settings for the GA strategy parameters a total of 92 different solutions were found, which performed better than the parameters from the official DELPHI tuning in 1992. The parameter values for the best-performing individual are listed in table G4.1.2, together with their associated χ^2 values.

The overall χ^2, value as well as the χ^2 values for each considered distribution, could be improved significantly. For a discussion of the results for the individual distributions, see Hahn *et al* (1994). On average, about 380 function evaluations were needed to find a better-performing solution than that of the DELPHI tuning in 1992, where approximately 250 iterations were needed for the tuning of four parameters. (Here, the Lund a parameter was not tuned but fixed to an arbitrary value since the applied algorithm was not capable of tuning both Lund parameters a and b.) Considering also the saved computation time due to the approximate evaluation of poorer-performing individuals, the GA seems to perform quite better than the standard method. Indeed, the main advantage of the GA is the richness of additional information, which will be illustrated in the following section G4.1.5.

Table G4.1.2. Optimized JETSET 7.3 PS parameters and associated χ^2 values. Listed are the values for the DELPHI tuning in 1992 and the best parameter set found by the GA. Note that for the DELPHI tuning the Lund a parameter was not tuned but fixed to an arbitrary value.

Parameter	1992 DELPHI tuning	GA optimized values
Lund a	0.180	0.276
Lund b	0.340	0.585
σ_q	0.395	0.368
Q_0	1.300	1.504
Λ_{LLA}	0.255	0.341
χ^2 / d.f.	3.79	1.90

G4.1.5 Analyzing the structure of the parameter space

The fact that we have so many different solutions is really surprising and suggests further investigations. Figure G4.1.3 shows the distributions of the parameter σ_q and the absolute difference of the Lund parameters $|a - b|$. Shown are the parameter values only of those individuals which where considered to be 'good' solution.

Both distributions reveal clearly the multimodal structure of the parameter space. One can observe solutions with the difference $|a - b|$ spread over a wide range of values, whereas there are just two distinct ranges for the σ_q parameter. For both parameters, the individuals with the highest fitness values are to be found in the region near the default values of the MC program, whereas the individuals with parameter values close to the values of the 1992 DELPHI tuning have smaller fitness values, indicating that this region might not be that of the global optimum. This observation might, of course, be an artifact of the search procedure, which might just miss those points in the five-dimensional space belonging to the real global optimum. Here one must keep in mind that the fraction of search points which have been evaluated is very small compared to the whole search space.

A further analysis is therefore required in order to find more arguments that the GA really encountered the region of the global optimum. For this purpose, the distribution of the σ_q parameter looks quite interesting. Here we have two distinct regions where the individuals with the highest fitness belong to the region with the smaller σ_q values. This parameter, which describes the width of the distribution of the transverse momenta of fragmenting hadrons should, to first order, be independent of the center-of-mass

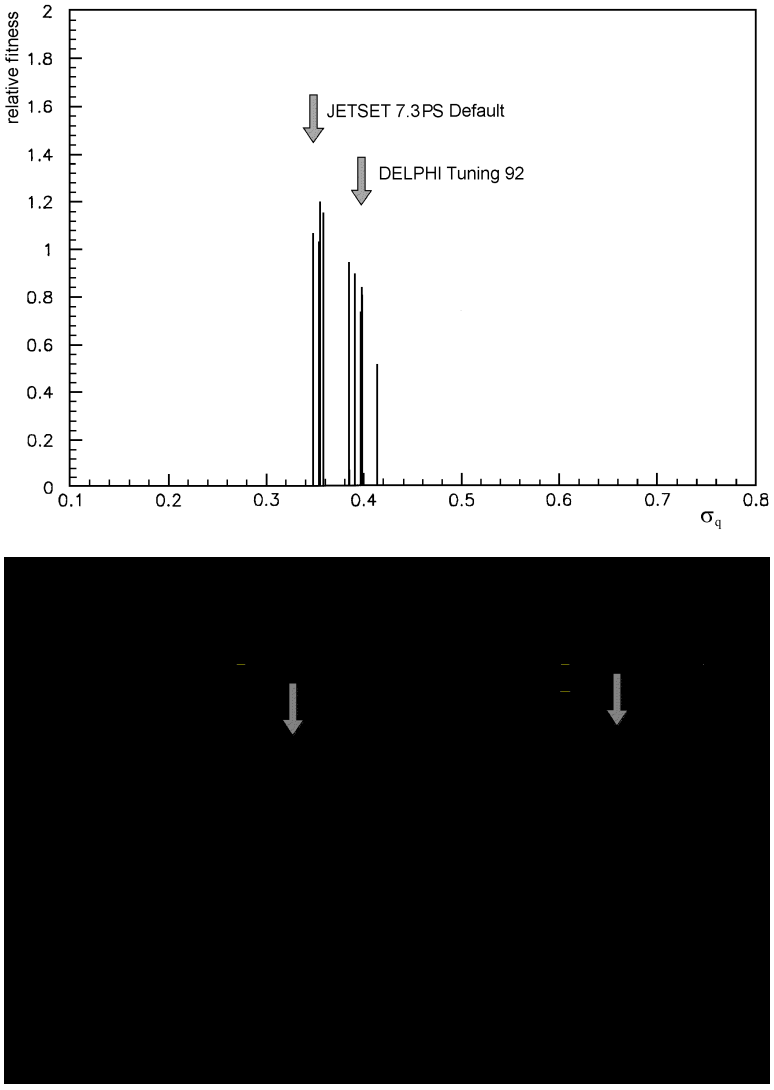


Figure G4.1.3. Distribution of the parameter σ_q (upper part) and the absolute difference of the Lund parameters $|a - b|$ (lower part) in the whole tuned interval. Shown are the parameter values only for individuals performing better than the parameters of the 1992 DELPHI tuning. In this and in the following figures the scale on the ordinate gives the fitness values associated with the specific parameter sets. Additionally, the parameter values of the 1992 DELPHI tuning and the default values of the MC generator are indicated.

energy. Therefore, our results agree more with the default values of the MC generator, which originate from a fit to lower-energy data.

In order to decide which of these regions is the right one, we also look at correlations between σ_q and other model parameters. Figure G4.1.4 shows correlations between σ_q and Q_0 and between σ_q and $|a-b|$. For both plots we see a broad region of solutions for high σ_q values and a narrow one for the small values. The same effect can also be observed for correlations between σ_q and other model parameters.

If we now look at correlations between other parameters, such as Q_0 and $|a - b|$ (figure G4.1.5) or between Λ_{LLA} and Q_0 (figure G4.1.6), we discover two relatively broad regions of combinations between these parameters for the whole range of σ_q (see parts (*a*)). If we only consider these correlations for the region of high σ_q values (parts (*b*)) the plots are nearly identical. But for the region with small σ_q values (parts (*c*)), we can observe a small, unique region of solutions. This clearly reveals that the region of small σ_q values belongs to the global optimum, whereas the solutions with high σ_q values (e.g. the parameters of the 1992 DELPHI tuning) belong to a suboptimal region of the search space.

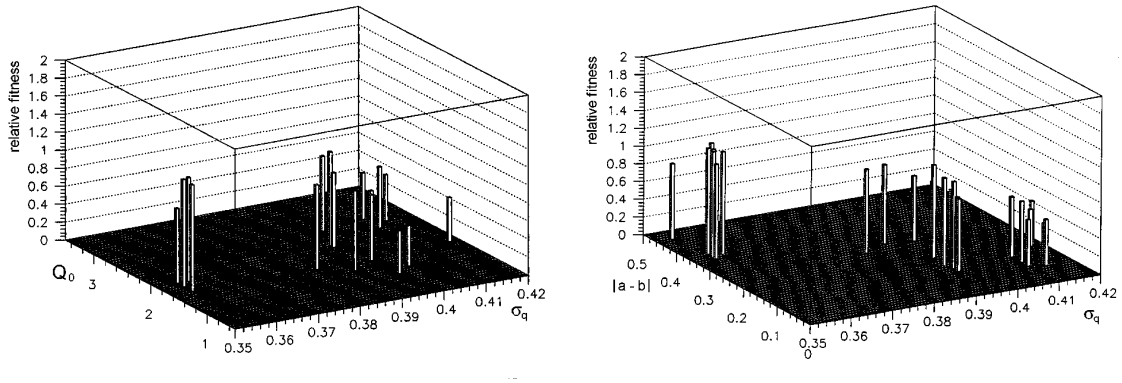


Figure G4.1.4. Correlations between the parameters σ_q and Q_0 (left) and between σ_q and $|a-b|$ (right).

G4.1.6 Conclusions

We have sucessfully used a genetic algorithm for tuning Monte Carlo generator parameters. The GA offers an automated procedure for this optimization task which does not require problem-specific expert knowledge.

A major challenge for the algorithm is the statistical fluctuation of the fitness function. Because of the extensive CPU time consumption the number of function evaluations has to be restricted to a minimum. A standard GA is not able to find an appropriate solution within a reasonable time limit. The implementation of a sharing model helps to maintain the populations' diversity and reinforces the global character of the search procedure. The limited resolution of the fitness function can be taken into account by carefully choosing the number of allocated bits for the coding of each parameter.

Due to the global character of the search strategy, the whole range of possible parameter values can be

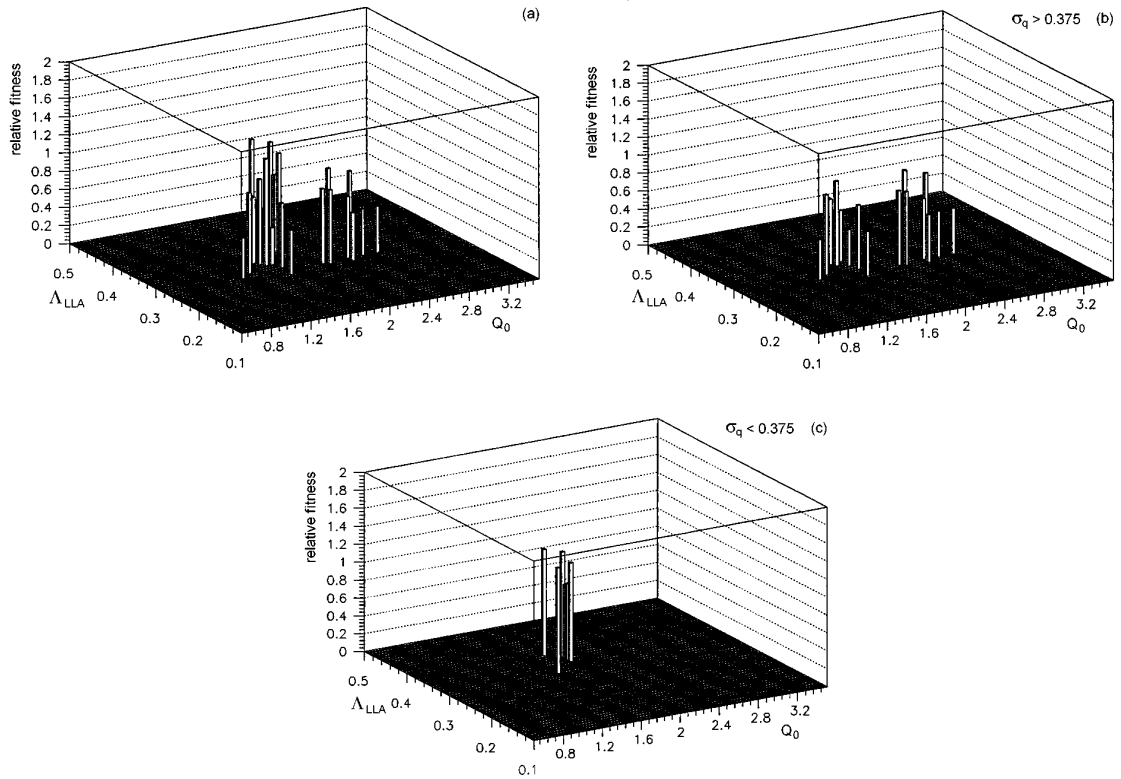


Figure G4.1.5. Correlation between the parameter Q_0 and the difference $|a-b|$. In (*a*) the correlation is shown for the whole σ_q range, in (*b*) the correlation is shown for $\sigma_q > 0.375$ and in (*c*) for $\sigma_q < 0.375$.

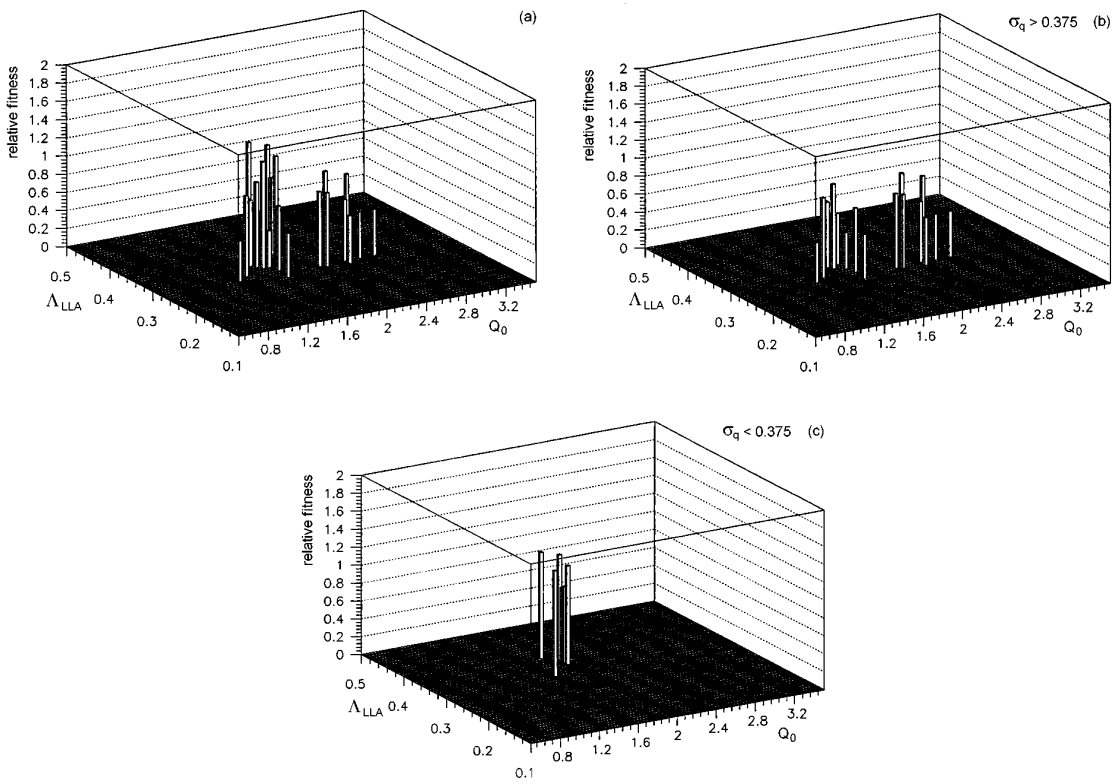


Figure G4.1.6. The same plots as in figure G4.1.5, but here for the correlation of the parameters Λ_{LLA} and Q_0.

incorporated into the search. The GA finds many different solutions and reveals the multimodal structure of the search space. Inspection of the parameters and their correlations yields additional useful information and allows the identification of the global optimum.

Considering its performance, a GA is competitive with standard optimization procedures. Partial, approximate evaluation of worse-performing individuals can significantly reduce computation time. Due to the inherent parallelism of the strategy, a GA should be especially effective in optimizing models that contain many free parameters.

Conventional optimization procedures always run the risk of subjective criteria influencing the results of the search, since on the basis of previous knowledge the human expert already has an idea of what the results should be. In contrast, the decision mechanism of a GA, which uses the principles of evolution, can be regarded as almost objective.

References

ALEPH Collab., Buskulic D *et al* 1992 Properties of hadronic Z decays and test of QCD generators *Zeitschrift für Physik* C **55** 209

Altarelli G *et al* (ed) 1989 *Z Physics at LEP I, Vol 1: Standard Physics (Proceedings of the Workshop on Z Physics at LEP I)* CERN Report 89-08

Bambah B *et al* 1989 *QCD Generators for LEP* CERN-TH. 5466/89

DELPHI Collab., P Aarnio *et al* 1990 Study of hadronic decays of the Z^0 boson *Phys. Lett.* **B 241** 271

Fürstenau H 1993 *Corrected Data Distributions and Monte Carlo Tuning* DELPHI Note 93-17 Phys 265

Goldberg D E 1989 *Genetic Algorithms in Search, Optimization and Machine Learning* (Reading, MA: Addison-Wesley)

Goldberg D E, Deb K and Clark J H 1991 *Genetic Algorithms, Noise, and the Sizing of Populations* IlliGAL Report No 91010, University of Illinois at Urbana-Champaign

Goldberg D E and Richardson J 1987 Genetic algorithms with sharing for multimodal function optimization *Proc. 2nd Int. Conf. on Genetic Algorithms (Cambridge, MA, 1987)* ed J J Grefenstette (Hillsdale, NJ: Erlbaum)

Hahn S 1993 *Optimierung von Monte Carlo Generator Parametern mit Genetischen Algorithmen* Diploma thesis WU D 93-24, University of Wuppertal

Hahn S, Becks K H and Hemker A 1992 Optimizing Monte Carlo generator parameters using genetic algorithms *New Computing Techniques in Physics Research II (Proceedings of the Second International Workshop on Software Engeneering, Artificial Intelligence and Expert Systems in High Energy and Nuclear Physics)* ed D Perret-Gallix (Singapore: World Scientific) pp 255–65

——1994 Solving optimization problems using evolutionary algorithms *New Computing Techniques in Physics Research III (Proc. 3rd Int. Workshop on Software Engineering, Artificial Intelligence and Expert Systems in High Energy and Nuclear Physics)* ed D Perret-Gallix and K H Becks (Singapore: World Scientific) pp 241–52

James J 1994 *MINUIT, Function Minimization and Error Analysis, Version 94.1, Reference Manual* CERN Program Library Long Writeup D 506

Sjöstrand T 1989 QCD Generators *Z Physics at LEP I, Vol 3: Event Generators and Software (Proceedings of the Workshop on Z Physics at LEP I)* CERN Report 89-08

Sjöstrand T 1992 *PHYTIA 5.6 and JETSET 7.3, Physics and Manual* CERN-TH. 6488/92

G4.2 Design optimization of a linear accelerator using evolution strategy: solving a TSP-like optimization problem

Hans-Georg Beyer

Abstract

The application of the evolution strategy (ES) for the approximative solution of a large-scale ordering problem appearing in the design studies of a LINAC (linear accelerator) in high-energy physics is discussed. The objective is to minimize the multibunch beam breakup (mbBBU) by finding a suitable ordering of the accelerator structures. This has been done by optimization of a computer model which simulates the behavior of the LINAC. From the evolutionary algorithms tested (genetic algorithm with nonelitist selection, (μ, λ) ES and $(\mu + \lambda)$ ES), the $(\mu + \lambda)$ ES has proven to be the most efficacious. An explanation for this result is provided which is in accordance with theoretical results on real-valued parameter optimization problems. The implementation of the ES on different parallel computer architectures using the master–worker paradigm is also discussed.

G4.2.1 Overview

An important problem in high-energy linear accelerators is the minimization of the multibunch beam breakup (mbBBU).

This case study provides an approach considered and performed during a design study for a 500 GeV S-band linear collider (Balewski *et al* 1991). The idea was to minimize the mbBBU by finding an optimal order of the accelerator components (especially, the ordering of the RF accelerator cavities is of interest). This work is organized as follows. A short description of the mbBBU phenomenon will be given. Knowing the working mechanism, it is possible to develop suitable methods that allow for a reduction of the mbBBU. The reason for the chosen approach will be explained. The simulation model is then introduced as a black box. Within this box there is a so-called 'tracking code' that simulates the entire collider. The development of the tracking code was not the subject of the project. The main body of this contribution deals with the evolutionary algorithms (EAs) used. Different EAs have been tested. The $(\mu + \lambda)$ *evolution strategy* (ES) B1.3 has proved to be the best variant. All details of this ES will be presented.

The first implementation of the ES was on a Sun workstation. It took the author *less than one week* to write and test the ES as well as the interface routines for the tracking code. Furthermore, the $(\mu \dot{+} \lambda)$ ES, which was implemented first, worked immediately. The results and experiences from further tests on ESs and *genetic algorithms* (GAs) will be also discussed. B1.2

Due to this very fast first success, the author parallelized the population. Because the tracking simulations for one offspring had taken up to 1 minute and the problem size is comparable with the complexity of a *traveling salesman problem* (TSP) with more than 2200 cities, this decision was quite G9.5 natural. An first attempt was on a Sun cluster connected with a LAN. It took only two weeks to succeed with the parallel version. Later on, other implementations on transputer-based parallel computers and on a VSM (virtual shared memory) KSR1 computer were successfully realized. Some experiences from these implementations will be reported.

G4.2.2 Problem description

G4.2.2.1 The multibunch beam breakup problem

The mbBBU is a serious problem to be coped with in the design phase of future TeV e^+e^- linear colliders (these are special linear accelerators, short LINACs, which accelerate positrons e^+ and electrons e^- separately in order to collide them).

For our considerations (only!) a LINAC can be simply imagined as a device consisting of a linear arrangement of metallic cavities (in the proposal of Balewski *et al* (1991) 2200–2500 cavities, about 15 km long), through which the electrons and positrons, concentrated in so-called bunches (one bunch contains up to 2×10^{10} particles; the bunch to bunch distance is about 3 m in the proposal cited above), are to be accelerated by feeding the cavities with electrical radiofrequency (RF) power. This accelerator method would work well if there were not an interaction of the charged bunches with the cavities called the *wakefield effect*. In order to have a high collision rate (in accelerator physics notation, *luminosity*) of the bunches at the interaction point it is necessary to guarantee the bunch position and the transverse beam dimension at a certain level (the beam dimension is in the range of 0.01–1 μm). Due to the wakefield effects, the required transverse beam dimension is difficult to guarantee: The first bunches induce wakefields acting back on the following bunches in terms of transverse kicks driving the bunches off axis. Since the induced wakefields increase with the off-axis distance of the bunches, initially small offsets grow exponentially and produce the so-called beam breakup.

G4.2.2.2 Beam breakup minimization approaches

Because the beam breakup (BBU) is a collective and cumulative effect, a remedy is to destroy the collective excitation of the wakefields, thus the bunches receive their kicks in a more or less random fashion with, hopefully, zero expectation (so-called Landau damping). Since the excitation of the wakefields depends upon the frequencies of the transverse modes of the cavities, to destroy the collective excitation one has to *detune* the cavity frequencies, for example, by mechanical deformation.

A better way than this *a posteriori* deformation technique applied to the material objects may be a *preventive* method using simulations of the entire LINAC. Such simulation programs—called tracking codes—are fed on the LINAC data (e.g. the cavity eigenfrequencies) and calculate the blow-up value D of the beam which is the measure of the BBU. Given such an objective function, there are two ways to minimize the BBU by simulation.

- Find the optimum cavity frequencies which minimize the blow-up factor. The frequencies are taken from a discrete set of 20 to 100 elements. This is a *discrete optimization problem*.
- The cavity frequencies are given. Ask for the optimum arrangement of the cavities which minimizes the blow-up factor, that is, solve the *order problem*.

Both procedures have been investigated by appropriated ESs. The first, the discrete optimization problem, can be easily implemented using the $(\mu \dot{+} \lambda)$ ES (Schwefel 1995) with discrete mutations and restriction of the search domain. Unfortunately, from the technological point of view it is difficult to produce 2000–3000 cavities in such a high precision (with respect to their eigenfrequencies) guaranteeing a one-to-one correspondence to the simulation.

The second approach is more reliable. One produces, for instance 20–100 classes of cavities having a natural frequency spread, measures the actual frequencies of the cavities, and asks how to assemble them in order to obtain a minimum BBU value D. This second method will be explained in detail.

G4.2.2.3 The simulation model to be optimized

In order to formulate the optimization problem it is assumed that there is a tracking code (a simulation program) *acting as a black box* producing an output D which is a measure for the blowup of the beam at the end of the LINAC. From the mathematical point of view we have a function

$$D = D(f_1, f_2, \ldots, f_p, \ldots, f_n) = D(\boldsymbol{f}) \tag{G4.2.1}$$

to be minimized depending upon n variables or vectors (in our case $n = 2200$–2500). Each position in (G4.2.1) describes a fixed cavity position p in the LINAC, for example, the first cavity at the start of the LINAC corresponds to the first position in (G4.2.1), and so on. The properties of a certain cavity $\#\pi$

(read 'number π') may be given by F_π (F_π contains the actual eigenmode frequencies of the cavity and perhaps other relevant cavity parameters). Thus, for example, let $f_2 := F_{1017}$ means put the cavity #1017 at position 2 in the LINAC.

It is assumed that the number of cavities is Π with $\Pi \geq n$, that is, there are at least n cavities, otherwise the LINAC could not be built. The case $\Pi > n$ expresses the (theoretical) possibility that there are more cavities produced than really needed. This is some kind of provision that allows for a faster evolutionary progress because of the extension of the search space (not used here, for details see Beyer 1992).

If we take a certain cavity arrangement $\boldsymbol{f}$, for example, $\boldsymbol{f}_{(0)} = (F_1, F_2, \ldots, F_\pi, \ldots, F_n)$, we will obtain the blowup value $D_{(0)} = D(\boldsymbol{f}_{(0)})$. It is unlikely that this arrangement is the optimum constellation. The mathematical objective is therefore to find a cavity order $\hat{\boldsymbol{a}}$, $\hat{\boldsymbol{a}} = (\hat{a}_1, \hat{a}_2, \ldots, \hat{a}_\pi, \ldots, \hat{a}_n)$, that is, $\boldsymbol{f}_{(\mathrm{opt})} = (F_{\hat{a}_1}, F_{\hat{a}_2}, \ldots, F_{\hat{a}_\pi}, \ldots, F_{\hat{a}_n})$ which minimizes the blowup value D, $\hat{D} = D(\boldsymbol{f}_{(\mathrm{opt})}) \stackrel{!}{=} \mathrm{Min}$. In other words, we have to solve an *order problem*.

The similarity to the TSP (traveling salesman problem) is striking, provided that $\Pi = n$ holds. The main difference is the objective D, instead of the tour length the more complex function $D = D(\boldsymbol{a})$ is to determine.

There is a second difference stemming from the tracking code used: the minimization of subpaths, greedy algorithms†, and the like are excluded. The LINAC is treated as a black box; we have something similar to a *blind* TSP with $n = 2200$–2500 cities. Due to the complexity of the problem to be solved we are aiming at 'good' solutions guaranteeing a reliable performance of the LINAC, and not particularly at (almost) optimum solutions, as usually demanded by the 'TSP community'.

G4.2.3 The evolution strategy used

G4.2.3.1 General aspects

Though ESs have been used mainly for real-valued parameter optimization, they can be applied to combinatorial problems as well. In this project, (μ, λ) and $(\mu+\lambda)$ ESs were tested without recombination. The main reason for this restriction was due to the lack of a theory, which gives a plausible explanation for the benefits of recombination (see, however, Section B2.4.1), and the disappointing results obtained B2.4.1 from GA implementations (see below).

Since a formal definition of $(\mu \stackrel{+}{,} \lambda)$ ESs on real-valued parameter optimization is provided in Section B1.3 a rather informal but detailed description of the ES on the order problem is presented here, B1.3 with comments enclosed in curly brackets.

(i) `Initialization`

(a) generate μ parental cavity arrangements $\boldsymbol{a}_{\mathrm{p}}^m$ ($m = 1 \ldots \mu$); in the simplest case choose $\boldsymbol{a}_{\mathrm{p}}^1 := \boldsymbol{a}_{\mathrm{p}}^2 := \cdots := \boldsymbol{a}_{\mathrm{p}}^\mu := (1, 2, 3, \ldots, \Pi)$

(b) compute the blowup values $D_{\mathrm{p}}^m := D(\boldsymbol{a}_{\mathrm{p}}^m)$ {*the fitness, obtained by the tracking code*}

(c) generate μ parental strategy parameters $s_{\mathrm{p}}^m := n/5$

{*beginning of the evolution loop*}

(ii) `Reproduction`

produce λ ($\lambda > \mu$) offspring $(\boldsymbol{a}^l, s^l)$

each offspring is produced by random choice of the parent number

$k := \mathtt{Random}\{1, \ldots, \mu\}$

and inheritance of the parental states

$\boldsymbol{a}^l := \boldsymbol{a}_{\mathrm{p}}^k$; $s^l := s_{\mathrm{p}}^k$

(iii) `Mutation`

for each offspring {*denoted by index 'o'*} do

(a) first, mutate the strategy parameter {*see section G4.2.3.2*}

$s_{\mathrm{o}}^l := \mathtt{mutate}(s^l)$

(b) mutate the cavity arrangement {*see section G4.2.3.2*}

$\boldsymbol{a}_{\mathrm{o}}^l := \mathtt{mutate}\left(\boldsymbol{a}^l, \left[s_{\mathrm{o}}^l\right] + 1\right)$

† For example, an often-used greedy technique in TSPs is to take the nearest neighbor to build the tour. Note that this technique provides in most cases only suboptimal solutions.

(iv) `Fitness evaluation and selection`

(a) apply the tracking code to each of the λ offspring and determine their blowup values $D_{\mathrm{o}}^{l} := D(\boldsymbol{a}_{\mathrm{o}}^{l})$

(b) selection: produce the new μ parents $\boldsymbol{a}_{\mathrm{p}}^{m}$

alternatively $(\mu + \lambda)$ or (μ, λ) selection:

$(\mu + \lambda)$ selection

select the μ best individuals from both the parents *and* the offspring according to their blowup values D

$(\boldsymbol{a}_{\mathrm{p}}^{1}, s_{\mathrm{p}}^{1}), \ldots, (\boldsymbol{a}_{\mathrm{p}}^{\mu}, s_{\mathrm{p}}^{\mu}) := \mathtt{selection}\big((\boldsymbol{a}_{\mathrm{p}}^{1}, s_{\mathrm{p}}^{1}), \ldots, (\boldsymbol{a}_{\mathrm{p}}^{\mu}, s_{\mathrm{p}}^{\mu}), (\boldsymbol{a}_{\mathrm{o}}^{1}, s_{\mathrm{o}}^{1}), \ldots, (\boldsymbol{a}_{\mathrm{o}}^{\lambda}, s_{\mathrm{o}}^{\lambda})\big)$

alternatively:

(μ, λ) selection

select the μ best individuals from the offspring according to their blowup values D

$(\boldsymbol{a}_{\mathrm{p}}^{1}, s_{\mathrm{p}}^{1}), \ldots, (\boldsymbol{a}_{\mathrm{p}}^{\mu}, s_{\mathrm{p}}^{\mu}) := \mathtt{selection}\,\big((\boldsymbol{a}_{\mathrm{o}}^{1}, s_{\mathrm{o}}^{1}), \ldots, (\boldsymbol{a}_{\mathrm{o}}^{\lambda}, s_{\mathrm{o}}^{\lambda})\big)$

(v) `Stop criterion`

if stopping criterion *not* fulfilled `GO TO (ii)`

Note that the algorithm implements *self-adaptation*. Therefore, each individual consists of the arrangement vector $\boldsymbol{a}$ (which describes the ordering of the cavities) and a strategy parameter s; s in entirety, denoted by $[s]$, is a measure of the 'mutation strength' applied to that individual. The details will be discussed next in section G4.2.3.2. C7.1

G4.2.3.2 Some special details

In order to understand the notion of a self-adaptive 'mutation strength', we first have to define how to mutate the $\boldsymbol{a}$-vector. The $\boldsymbol{a}$-vector contains the cavity order (cf section G4.2.2.3). Each permutation of this ordering gives a new feasible cavity order. The smallest move step in this model is therefore given by the exchange of two cavities. That is, two positions i and j in $\boldsymbol{a} = (a_1, \ldots, a_\Pi)$ are chosen at random and their contents a_i and a_j are exchanged $a_i \leftrightarrow a_j$. This may be called a *2-exchange move*. The r-times iterated application of the 2-exchange move builds up a mutation of *mutation strength* r

$$\mathtt{mutate}(\boldsymbol{a}, r) := \mathtt{perform\ iteratively}\ r\ \mathtt{2\text{-}exchange\ moves\ on}\ \boldsymbol{a}.$$

In order to self-adapt the r-value, a real-valued strategy parameter s has been introduced

$$r := [s] + 1 \qquad s > 0. \tag{G4.2.2}$$

The '1' in equation (G4.2.2) ensures that there is at least one 2-exchange per offspring. $[s_{\mathrm{o}}^{l}]$ is the entire part of the strategy parameter s_{o}^{l} produced from the parental s^{l} by multiplicative mutation

$$s_{\mathrm{o}}^{l} := \mathtt{mutate}(s^{l}) = s^{l} \times \exp(z) \tag{G4.2.3}$$

z is an $N(0, \tau^2)$ normally distributed random number, thus s_{o}^{l} is a log-normally distributed variate. This method has been adopted from Schwefel (1995). Each parent has its own s-value to be left to its offspring. Since the best offspring are selected together with their mutation strength the population can learn the optimum mutation strength. The initial s-values should be of the order n ($n/5$ used), the number of cavity positions in D (cf equation (G4.2.1)). That is, at the start on average almost all cavity positions will be exchanged. This corresponds to the large mutation strength to be chosen at the start in traditional function optimizing ES.

There are certain connections to simulated annealing. At the optimization start a Monte Carlo search is performed. This corresponds to a high temperature (s is therefore the counterpart to the temperature). However, in our case an annealing schedule is not needed, since the s-values are adapted by the evolution process. The only open parameter in this strategy is the standard deviation τ of the normally distributed random numbers z in equation (G4.2.3), being a measure of the fluctuation strength of s. A τ-value of $\tau = 0.4$ has proven to be a good compromise between a high rate of 'innovation' (τ high) and 'unteachableness' ($\tau = 0$).

An important question in all evolution-like methods is the choice of the strategy parameters. For the $(\mu \dot{+} \lambda)$ ES these are the number of parents μ and offspring λ.

For the first experiments, a $(5 + 12)$ ES was chosen, with the following arguments. (Subsequent implementations on parallel computers were also tested with higher offspring numbers.) Since the rate of progress (and also the quality gain) is a monotonically increasing function of the number of offspring one should use a high λ-number. Unfortunately the increase is degressive (for the so-called sphere model, it can be shown that the rate of progress rises asymptotically with the logarithm of λ). Therefore, raising the number of offspring much higher than the number of processors provides only a small performance gain.

How many parents should be chosen? Choosing only one would be a hard selection, guaranteeing the highest local progress rate, but with the disadvantage of a higher probability of becoming trapped in a local optimum, due to lack in variability of the genetic information. (Genetic information refers here to the $\boldsymbol{a}$-vector containing the ordering information. From the evolutionary programming (EP) point of view one might also interpret this as *phenotypic* information.) If we choose the number of parents too high the opposite would be the case—the population would move very slowly through the fitness landscape. Assuming four processors, the $(5 + 12)$ ES seems to be a useful compromise. However, if the number of processors is larger—say 20 or more—then λ should be equal to this number and the parent number μ should be chosen in the range of $\mu \approx 0.2$–$0.05\,\lambda$.

Concerning the stop criterion, the usual criteria can be applied. Since computer power is restricted, the maximum number of generations or the CPU time used has served as a stopping criterion in most of the runs.

G4.2.3.3 Parallel implementations

The optimization task to be performed is almost ideal for parallelization. First, due to the $\{\lambda, \mu\}$ population structure there is a 'top-level' parallelism inherent in the ES which is well suited for MIMD machines. The parallelization paradigm used is the so-called *master–worker principle* (this is sometimes referred to as the *master–slave* or *farmer–worker* principle) which provides a 'coarse-grained parallelism' with easy to implement properties. Second, the fitness computation takes a very long time (almost 40 seconds for one offspring on a SPARC 2 workstation). Therefore, it is quite clear that there will be no communication bottleneck which usually affects the performance of master–worker implementations.

The $(\mu \stackrel{+}{,} \lambda)$ ES has been implemented in a 'trivial' fashion. The whole $(\mu \stackrel{+}{,} \lambda)$ algorithm runs on the master with the exception of the time-consuming fitness calculations which are done on the workers.

The first parallel version of this ES was realized on a workstation cluster without any special communication software. This was in spring 1991. Today one would use special message passing libraries such as the public domain package PVM (parallel virtual machine) or the commercial products EXPRESS or PARMACS. However, in 1991 such communication software was not available for the author. Therefore, the following approach was tried on a Sun cluster connected by Ethernet LAN using only standard Sun-Fortran (Beyer 1992).

The cluster consists of four Sun-SPARC stations with an NFS (shared network filesystem service). One host works as an 'evolution master' performing the reproduction of the parents and making the selection. All hosts run a process which may be called a 'brooder'. These brooders obtain their 'eggs' from the evolution master; the egg, containing the $\boldsymbol{a}$-vector describing the cavity arrangement of one offspring, is put in a file called 'gamete' to be read from the brooder. The brooder performs the time-consuming tracking calculations yielding the blowup factor D which is put in a common file called 'fitness' to be read by the evolution master. Each brooder has its own 'gamete' file, but there is only one 'fitness' file for the whole LAN. Experiments have shown that it is necessary to use for each brooder an own gamete instead of a common file, since the extensive read and write accesses at the beginning of a new generation seem to confuse the system and destroys the file consistency (there is no direct way to lock a file within Sun-Fortran). This problem does not hold for the 'fitness' file, since it is very short (only about 20–40 bytes) and the probability of accessing it from different brooders at the same time is very small.

This simple approach did work so reliably that a more elegant communication technique, such as socket-based process communication, was never tried. The system has an astonishingly high fault tolerance as long as the evolution master is not affected. This has been achieved by a simple time-out mechanism in the master which assumes the brooder to be 'currently dead' if the brooder does not return a fitness value within a time period (in the special case 10 minutes). In such a case the brooder's egg is given to another idle brooder.

Note that this approach exhibits a natural and efficient *load balancing*: processors/hosts with heavy

load 'breed their eggs' slowly (the ES program runs in the background with low priority), whereas those workers with low load produce on average more offspring. This self-organizing process works best (on average) if the number of offspring λ is a sufficiently large multiple (≥ 3) of the number of processors.

Because of the very fast success of this 'Sun parallel computer', further implementations on real parallel architectures have been tested. The first was a Parsytec 32-T800 transputer system with a HELIOS and later on a PARIX operating system. Using the message passing routines of the operating systems, the basic master–worker concept from the workstation cluster could be easily transfered to the transputer system. Unlike the four-processor Sun version realizing a $(5 + 12)$ ES, $(\mu \stackrel{+}{,} \lambda)$ strategies with $\lambda \leq 31$ have been tested on the 32-transputer system. That is, each processor obtains only one offspring per generation. Load balancing is not possible or necessary because only a single user application can run on the processor.

The scaling behavior of this application exhibits a linear speedup with the number of processors involved (concerning the number of offspring computed within a certain time). This does not come as a surprise because the communication time per processor was only $1/1000$ of the computing time. Therefore, it would be (theoretically) possible to raise the number of processors/offspring λ by a factor of ten without significant performance degradation.

A further parallel implementation has been made on a KSR1 computer with VSM (virtual shared memory) architecture. Unlike the message passing paradigm where the programmer has to ensure the data consistency by software, on the KSR1 computer data consistency is guaranteed by special hardware. Though there is a cache hierarchy, called ALLCACHE (Frank *et al* 1993), which is local to the processors, the user is provided with a uniform address space for instructions and data. Thus, the result of program execution on the multiprocessor system is equivalent to the execution of the program on a single processor.

Taking advantage of the benefits of this VSM system requires some changes in the worker as well as the master code. Actually, master and worker are to integrate into a single program which starts the worker subroutines at the level of the so-called *pthread* (for details, see Beyer 1993). As a result, a code has been obtained that might also be used for parallel TSP optimizations, because of the fast communication.

G4.2.4 Results and conclusions

G4.2.4.1 Why is the $(\mu + \lambda)$ evolution strategy strongly recommended?

The $(\mu + \lambda)$ ES has proved to be the best EA algorithm tested on the problem of interest (including (μ, λ) ES and GAs). Due to the relatively high selection pressure of the $(+)$ elitism, no stagnation in the evolution process was observed. A typical evolution record for a $\Pi = n = 2228$ cavities LINAC is depicted in figure G4.2.1.

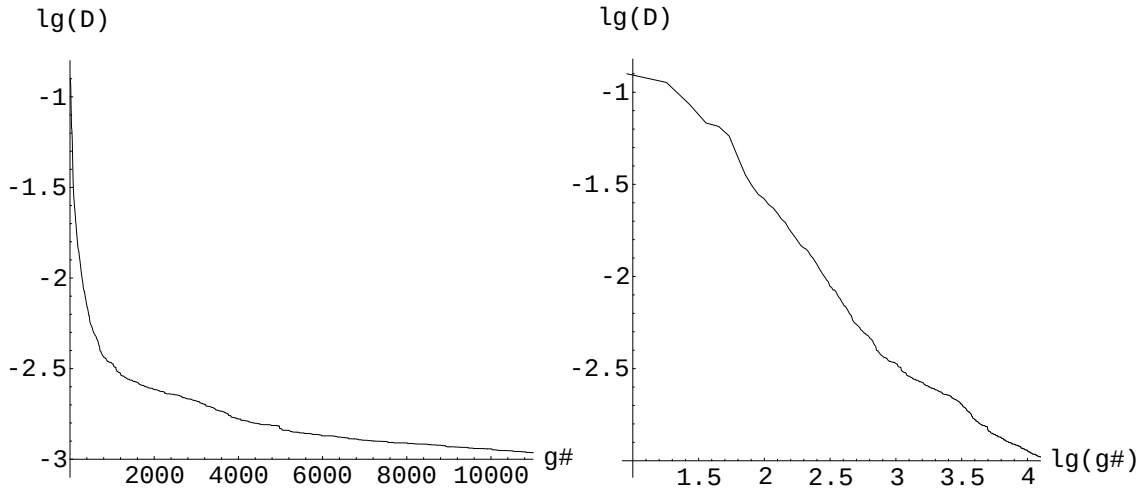


Figure G4.2.1. The dynamics of the $(5 + 12)$ ES 2228 cavity BBU minimization. The left graph shows the logarithm of the blowup value D versus the number of generations g. The right graph shows $\log(D)$ versus the logarithm of the generation number.

As can be seen, it takes the $(5 + 12)$ ES $g = 10\,000$ generations to decrease the blowup value by a factor of 100. If the graph in the right picture is extrapolated to the right, then one can estimate that a further decrease by a factor of ten (i.e. $\log(D) = -4$) would take roughly 100 times the generations needed to reach $\log(D) = -3$, that is $g = 1\,000\,000$ generations. For the time being, however, such improvements are excluded, even for highly parallel systems.

The graphs in figure G4.2.1 exhibit a remarkable behavior. From the ES theory on real-valued parameter optimization one would expect a linear convergence order. This corresponds to a (roughly) linear graph in the left picture of figure G4.2.1. However, for the combinatorial BBU problem it rather seems that $D \propto g^{-\alpha}$ $(\alpha > 0)$ does hold. This corresponds to a *sublinear* convergence order, which is observed in real-valued parameter spaces for $(\mu + \lambda)$ strategies, if the mutation strength σ is *held constant*. The examination of the r-values, equation (G4.2.2), reveals that after about $g \approx 1000$ generations r is equal to one. That is, the minimal possible mutation strength with respect to the 2-exchange has been reached. There seems to be a strong correspondence:

$$r = \text{constant} \qquad \leftrightarrow \qquad \sigma = \text{constant}.$$

If this correspondence is correct, it becomes clear that (μ, λ) strategies are not well suited for combinatorial problems. Because the self-adaptation cannot produce smaller r than 1 (NB, $r = 0$ would mutate nothing) we have the case of constant mutation strength. It follows that in the real-valued parameter case such strategies exhibit a saturation. That is, they remain a certain distance from the optimum. If translated back to the combinatorial problem, one would expect a similar behavior. Indeed, this was observed and first reported by Beyer (1992). Therefore, (μ, λ) strategies on combinatorial problems cannot be globally convergent. However, if a very high offspring number λ is chosen, the saturation behavior can be shifted to equilibrium values that may be accepted as an approximation to the optimal solution. A theory for how to choose λ is up to now not available. Because of these factors *the use of* $(\mu + \lambda)$ *strategies is highly recommended* for combinatorial problems.

This recommendation also holds if GAs without elitism are taken into account. They have been tested (mbBBU and TSP minimization) without any satisfactory results. The evolution becomes stuck at a very early state, independent of the crossover techniques used. An increase of the population improves the results, that is, the equilibrium D-value shifts to smaller blowup values (very similar to the (μ, λ) ES), but the basic behavior of saturation remains. The main reason for this behavior of the nonelitist GA class is the random selection which prevents the conservation of good solutions found so far. If elitism is incorporated into the GA the results will become better.

G4.2.4.2 Concluding remarks

This contribution has dealt with the approximative solution of a large-scale order problem. Even though the finding of an optimal arrangement of accelerator cavities in a LINAC seems to be a very special problem of high-energy physics, the 'problem pattern' is very general, and there are surely many applications which are similar. If, for example, the 'cavities' are replaced by 'cities' one would obtain the classical TSP (but, change 2-exchange to Lin-2-opt to obtain the smallest mutation size possible!). In principle, each problem which tries to find an optimal ordering could be treated by the $(\mu + \lambda)$ ES presented.

Furthermore, if the fitness evaluation is rather time consuming, than the parallel versions are recommended. Due to the simple master–worker principle, parallelism can be easily implemented on each MIMD-like parallel machine or workstation cluster. Because of the coarse-grained parallelism, one can expect a linear speedup with the number of processors allocated. This scaling behavior concerns the number of offspring λ produced in a given time. One should not expect that the quality gain, that is, the average fitness change per generation, always rises linearly with λ. However, this is not an argument against a higher number of processors. If there are idle processors in the system they might as well be put to use by the $(\mu + \lambda)$ ES.

Acknowledgement

Part of this paper has been taken from the article by Beyer (1992) with kind permission from Elsevier Science BV, Amsterdam, The Netherlands.

References

Balewski K *et al* 1991 *Status Report of a 500 GeV S-Band Linear Collider Study* DESY Report 91-153

Beyer H-G 1992 Some aspects of the 'evolution strategy' for solving TSP-like optimization problems *Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature (Brussels, 1992)* ed R Männer and B Manderick (Amsterdam: Elsevier) pp 361–70

——1993 Optimization of large-scale order problems by the evolution strategy *One Year KSR1 at the University of Mannheim* ed R Schumacher University of Mannheim Computing Center, Technical Report RUM35/93, pp 11–6

Frank S, Burkhardt H III and Rothnie J 1993 The KSR1: high performance and ease of programming, no longer an oxymoron *Supercomputer '93, Anwendungen, Architekturen, Trends* ed H-W Meuer (Berlin: Springer) pp 53–70

Schwefel H-P 1995 *Evolution and Optimum Seeking* (New York: Wiley)

G4.3 Genetic programming for nonlinear equation fitting to chaotic data

E Howard N Oakley

Abstract

Current techniques for the investigation of chaos in data series require long, noise-free experimental measurements which are seldom available in biological and medical work. Genetic programming was seen to offer potential in a number of ways, and was therefore initially used to forecast future data values from very short and noisy input data. Genetic programming proved to be as effective a forecasting tool as others advocated in the literature. Forecasting error initially increased quickly with increasing length of prediction, then increased more slowly, according to a biphasic pattern described previously; the gradients of each limb may be used as a crude indicator of the sum of positive Lyapunov exponents. Although no S-expression ever exactly replicated that used to generate the data, fittest S-expressions did yield useful structural data. Furthermore, the efficacy of forecasting remained high even when noise was added to the data series. The application of genetic programming to original and surrogate data series may be a useful test between chaos and randomness. Runs on surrogate series failed to achieve the high fitness values seen with real data, and were distinguished by shallow and homogeneous populations of S-expressions.

G4.3.1 Project overview

There is considerable practical and theoretical interest in the investigation of systems which occur naturally and which may be chaotic in origin. A number of tools have been developed for this work, primarily from the physical rather than biological sciences: they include the estimation of various dimensions and Lyapunov exponents. These tools are most suited to the investigation of long, relatively noise free datasets, although biological, especially medical, datasets are much more commonly brief and contaminated with noise of an uncertain nature. Many biological time series are not stationary either. As a result, few studies of possible chaos in human physiology have progressed much beyond the estimation of simple measures, such as fractal dimension, and there is controversy over whether observations should be accounted for by chaotic or purely stochastic models. Some authorities, such as Glass and Kaplan (1994), appear happy to accept less rigorous evidence of chaos in high dimensions, whilst others, such as Ruelle (1990), emphasize that apparent high dimensionality may be a good indicator of stochastic systems.

In the course of experiments measuring the blood flow in the skin of healthy human volunteers, the present author became struck by the visual similarity between these data and mathematical models which are known to be chaotic, such as the Mackey–Glass series (Mackey and Glass 1977), and frustrated by the lack of tools which are suitable for further investigation of any such similarity. At the same time, the demonstration by Koza (1992) of the ability of *genetic programming* to perform local forecasting of the logistic function suggested that this might be a technique which was worth further investigation. This project was then undertaken by one person, intermittently over a period of two years to date. B1.5.1

The original aim of this project was to assess the efficacy of genetic programming as a technique for forecasting chaotic series, and, by measuring the success of such forecasts, to try to use them as a means

of classifying datasets in terms of chaos and randomness. However, it has become apparent most recently that the versatile and increasingly popular technique of analyzing surrogate data series might be coupled with forecasting by genetic programming to yield even more useful information.

By definition, chaotic data series cannot be forecast accurately, and as the forecast period is lengthened, so the inaccuracy of the forecast increases. A number of studies have used different local forecasting techniques to try to predict data series into the future. The methods employed have included radial basis functions, piecewise linear approximation, neural networks, and the *genetic algorithm* (Oakley 1994b). B1.2 The difficulty of this work is exemplified by the study of Stokbro and Umberger (1992), which used neural networks with weighted maps over training series of 500–5000 observations to attempt to predict just six steps into the future. Following a long history of efforts by Fogel and others using evolutionary methods, Meyer and Packard (1992) ingeniously employed the genetic algorithm on a series which had been embedded in phase space, and achieved success in forecasting very limited regions of attractors. Interesting though these approaches are, they hold little promise for short, noisy biological data.

In theory, genetic programming should be capable of discovering the function(s) underlying a nonlinear map, and so solve the inverse problem of Casdagli (1989): 'given a sequence of iterates, construct a nonlinear map that gives rise to them. This map would then be a candidate for a predictive model.' It was also Casdagli *et al* (1992) who first recognized the promise that genetic programming, as opposed to other applications of the genetic algorithm, held for solving the inverse problem.

Surrogate data series, in this context, are derivatives of the original, possibly chaotic dataset which have been manipulated so as to remove the sequential associations between members of the series which could make them chaotic, but which still retain the nonchaotic properties of the data series. Their generation and use has been described by Theiler *et al* (1992). Prediction using genetic programming could be applied to both original and surrogate series, and the results compared: if the data series does contain chaos, then there should be obvious differences between the two.

Genetic programming thus appeared to offer a wide range of valuable possibilities, ranging from system identification, through prediction, to formal testing for chaos.

G4.3.2 Design process

In the first phase of this study, genetic programming (Koza 1992) was applied to fit predictive *S-expressions* C1.6 to data from a known chaotic time series, the Mackey–Glass map. At this stage, it was considered valuable to ensure that it was possible to generate an S-expression which was a perfect fit (Koza's sufficiency criterion), so instead of using the original delay differential equation of Mackey and Glass (1977), the following map was employed:

$$x_{t+1} = x_t + \frac{bx_{t-\Delta}}{1 + x_{t-\Delta}{}^c} - ax_t$$

where x_t is the value of x at time t, a is 0.1, b is 0.2, c is 10.0, and Δ is 30.0. This equation is the discretized delay difference equivalent of the Mackey–Glass delay differential equation, rather than an attempt to approximate the differential form.

In the second phase, the data generated by this map were progressively contaminated with stochastic noise, including both additive and multiplicative components. Following this, predictions using genetic programming were performed on data generated from the original delay differential equation, and surrogates of this and the Mackey–Glass map. The current and final phase of this project is repeating these studies on a selection of physiological datasets (blood flow measurements in the skin of the great toe).

The length of data series has been deliberately constrained to parallel that achievable in many physiological and biological studies, typically 1065 exemplars and less. During genetic programming, members of the population of S-expressions have been even more constrained to just 35 real members of the series, and expected to predict from 10 to 100 steps into the future. This was repeated over seven to ten nonoverlapping sections of the complete data set, and the predicted and actual values compared. In a few experiments, single prediction runs were made over much longer periods into the future, to over 1000 steps, to assess asymptotic performance.

The raw fitness function is then simply the root mean square (rms) prediction error across all forecasts made by a given S-expression. For the purposes of making comparisons between different runs when the test data contained noise, this was normalized against the prediction error between the correct map and the noisy data, as used by Farmer and Sidorowich (1987).

In early studies, experiments were performed using different fitness functions, such as one based on comparisons between the Fourier power spectra of the predicted and real data series. These were also used in combination with simple rms error, in an effort to 'steer' the population first to generate a waveform similar to that seen in the original data, and then to tune that waveform until it was in phase.

Time delay embedding was effectively achieved by the inclusion in the terminal set of some of the 35 'seed' exemplars. In conventional embedding, knowledge of the dimensionality and the requisite delay are required before analysis can proceed. This is a problem with short data series because of the paucity of information, and it remains a more subjective step even in long, clean data series. However, the use of a number of differently spaced values within the terminal set could effectively assess the required embedding without *a priori* information: in this sense, genetic programming could attempt to perform elements of system identification at the same time as prediction.

A potential source of bias could be in the choice of terminal and function sets. Both were deliberately kept large, in order to allow genetically richer populations to develop as well as avoiding any bias. Some runs were performed which deliberately omitted members of the terminal and function sets which were required for sufficiency, that is, the S-expression required to form a perfect fit with the actual data series could not be directly expressed with the variables and operators provided.

G4.3.3 Development and implementation

Input datasets consisting of 1024–1065 values were generated using double-precision floating point mathematical routines according to the descriptions given above. In cases of the Mackey–Glass map and delay differential flow, the series was seeded with a pseudorandom sequence, and the first 1000 exemplars were discarded. Additionally, a 'random walk' data series was generated by using signed pseudorandom numbers as the increments to a single starting value.

Genetic programming was performed using the Simple Lisp implementation of Koza (1992), incorporating performance enhancements that were specific to the Common Lisp which was being used. These accelerated the evaluation of S-expressions within the population without loss of accuracy. Macintosh Common Lisp version 2.0.1 (Apple Computer) was employed on Apple Macintosh 68040 (Quadra 950 and IIci with Radius Rocket accelerator) computers. Exploratory series were undertaken initially to investigate the optimum settings for values within Koza's 'tableau', after which there were production series which typically took 24–72 hours for 10 to 20 runs on each occasion.

Genetic programming typically used input data values 1, 2, 3, 4, 5, 6, 11, 16, 21, and 31 time points prior to the start of prediction, together with generated random real numbers, as the terminal set. The function set typically included the four real arithmetic operators (+, −, ∗, and division protected from divide by zero errors), together with sine, cosine, and exponentiation to the power of 10 protected from overflow and underflow errors. Initial populations of 50–5000 S-expressions were generated using Koza's ramped half-and-half method, with a maximum depth of six. The selection method was fitness proportionate with reproduction fraction 0.1 and a maximum depth after crossover of 17; some runs were performed using tournament selection instead. Each run was performed for 51 or 101 generations, but none terminated because the number of 'hits' or fitness was high enough to meet predetermined criteria. Complete details of the settings used are given by Oakley (1994a, 1994b). No attempt was made to use the more recent technique of automatic function definition Koza (1994), although it is intended to investigate this shortly.

G4.3.4 Results

Genetic programming performed as well as the iterative radial basis functions of Casdagli (1989), at the prediction of the Mackey–Glass map and its original delay differential flow, and showed the same biphasic relationship with the length of forecast. The latter indicates that not only does the accuracy of prediction fall with the length of forecast, but up to about 60 points into the future the rate of fall of accuracy is high, and it then levels off as forecast lengths increase further. This appears almost characteristic of chaotic systems. The best normalized rms error achieved forecasting 30 steps ahead was 0.1596 (Casdagli achieving 0.1585), whilst that for 100 steps ahead was 0.9136 (Casdagli 0.990),where 0.0 indicates perfect accuracy, and 1.0 complete inaccuracy.

Study of individual fittest S-expressions confirmed that local forecasting could attain great accuracy, but that the majority of S-expressions failed to model the phasic pattern of the Mackey–Glass map.

Commonly, the S-expression either diverged quickly from the original data series, or quickly settled to a constant value. It was for this reason that experiments were performed using two fitness functions. Whilst these did generate some interesting results, the generation of Fourier power spectra was too computationally intensive to be used on a larger scale given the limited computer resources available. The attraction of such 'steered' fitness functions remains that they may first generate S-expressions which have the right periodic qualities, and then may tune them until they are in phase with the data which they are trying to forecast.

There was also ample useful information in terms of system identification. The terminal set member representing the time delay of 30 found in the Mackey–Glass map occured statistically significantly more frequently in both long- and short-term prediction runs (Oakley 1994a). However, it never appeared in the form $1/(x_{t-31})^{10}$ found in the map. The two prevalent gradients in a semilogarithmic plot of normalized prediction error against the length of forecast were estimated to be 0.08121 (length less than 60 steps) and 4.655×10^{-4} (length greater than 80 steps). These compare with and may relate to a computed metric entropy, that is, the sum of positive Lyapunov exponents, of approximately 0.01 for the delay differential form of the Mackey–Glass series (Meyer and Packard 1992).

A detailed account of the results from the use of noisy datasets has been given by Oakley (1994b). Although noise inevitably increases prediction error, genetic programming as a means of forecasting appears remarkably robust in the face of even quite substantial amounts of noise.

Striking differences were observed in the effectiveness of genetic programming to fit surrogate as against real data series. Whilst many runs using original data (from the Mackey–Glass flow and map, and experimental work) behaved normally in all respects, there was a marked tendency for random walk and surrogate series to fail to evolve fitter individuals. This was manifested by the early discovery of false optimal fits, which were invariably short and contained simple S-expressions, and quickly dominated the population. These S-expressions recurred in most runs on a given surrogate or random walk series. In contrast, the original chaotic series exhibited more diverse populations of more complex S-expressions, and found a wider variety of fits which normally continued to improve through later generations. For example, an original Mackey–Glass map predicted 60 steps into the future showed fittest S-expressions with standardized fitnesses ranging between 15.4 and 28.5, which were found at an average generation of seven (range 0–26). Its surrogate showed fittest standardized fitnesses ranging between 7.7 and 10.1, found at an average generation of four (range 1–9). These results suggest that this could form the basis of a test for chaos versus randomness.

G4.3.5 Conclusions

Although genetic programming failed to discover an S-expression for the Mackey–Glass map from data series generated by it, and thus did not solve Casdagli's inverse problem, it has yielded potentially useful information regarding system identification and the chaotic nature of data series. This accords with the experience of Iba *et al* (1993) using a compound technique which includes the genetic algorithm. Genetic programming is also pleasingly robust in the face of noise.

These results also invite consideration of two potential tests, which may be of use in trying to distinguish chaos from randomness: the first is the plot of forecasting error against length of forecast, and the second the comparison of runs on real and surrogate data series.

It is perhaps worth bearing in mind that, throughout this project, the number of data provided to genetic programming has been only just sufficient. Ruelle (1990) has proposed that the absolute minimum is given by $D \leq 2 \log_{10} N$, where D is the correlation dimension of the system, in the case of the Mackey–Glass flow approximately three, and N the minimum number of observations. If this holds good here, then the expected minimum series from which useful information could be extracted is 32.

Genetic programming thus appears an excellent technique for the investigation of these short, noisy data series, and merits ongoing study in this role.

References

Casdagli M 1989 Nonlinear prediction of chaotic time series *Physica* **35D** 335–56

Casdagli M, des Jardins D, Eubank S, Farmer J D, Gibson J and Theiler J 1992 Nonlinear modelling of chaotic time series: theory and applications *Applied Chaos* ed J H Kim and J Stringer (New York: Wiley–Interscience) pp 335–80

Farmer J D and Sidorowich J J 1987 Predicting chaotic time series *Phys. Rev. Lett.* **59** 845–8

Glass L and Kaplan D 1994 Complex dynamics in physiology and medicine *Time Series Prediction: Forecasting the Future and Understanding the Past* ed A S Weigend and N A Gershenfeld (Reading, MA: Addison-Wesley) pp 513–28

Iba H, Kurita T, de Garis H and Sato T 1993 System identification using structured genetic algorithms *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann)

Koza J R 1992 *Genetic Programming. On the Programming of Computers by Means of Natural Selection* (Cambridge, MA: MIT Press)

——1994 *Genetic Programming II: Automatic Discovery of Reusable Programs* (Cambridge, MA: MIT Press)

Mackey M C and Glass L 1977 Oscillation and chaos in physiological control systems *Science* **197** 287–89

Meyer T P and Packard N H 1992 Local forecasting of high-dimensional chaotic dynamics *Nonlinear Modelling and Forecasting* ed M Casdagli and S Eubank (Redwood City, CA: Addison-Wesley) pp 249–63

Oakley E H N 1994a Two scientific applications of genetic programming: stack filter and non-linear equation fitting to chaotic data *Advances in Genetic Programming* ed K E Kinnear (Cambridge, MA: MIT Press) pp 369–89

——1994b The application of genetic programming to the investigation of short, noisy, chaotic data series *Evolutionary Computing* (*Lecture Notes in Computer Science 865* ed T C Fogarty (Berlin: Springer) pp 320–32

Ruelle D 1990 Deterministic chaos: the science and the fiction *Proc. R. Soc.* A **427** 241–8

Stokbro L and Umberger D K 1992 Forecasting with weighted maps *Nonlinear Modelling and Forecasting* ed M Casdagli S Eubank (Redwood City, CA: Addison-Wesley) pp 73–93

Theiler J, Galdrikian B, Longtin A, Eubank S and Farmer J D 1992 Using surrogate data to detect nonlinearity in time series *Nonlinear Modelling and Forecasting* ed M Casdagli and S Eubank (Redwood City, CA: Addison-Wesley) pp 163–88

Further reading

1. Iba H, Kurita T, de Garis H and Sato T 1993 System identification using structured genetic algorithms *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann)

   An innovative system which is built around evolutionary techniques, used to study chaotic dynamics.

2. Meyer T P and Packard N H 1992 Local forecasting of high-dimensional chaotic dynamics *Nonlinear Modelling and Forecasting* ed M Casdagli and S Eubank (Redwood City, CA: Addison-Wesley) pp 249–63

   An ingenious study which uses the genetic algorithm with time delay embedded data from a chaotic series, to produce local forecasts of great accuracy.

G5.1 Genetic algorithms for the analysis of the movement of airborne pollution

Hugh M Cartwright

Abstract

A genetic algorithm (GA) has been used to tackle the source apportionment problem—the allocation to individual sources of the pollution arriving at monitoring points in an urban area. This problem is environmentally important and computationally complex, but conventional methods of solution have met with limited success. We discuss here a variant on the standard GA, in which the algorithm is adapted to operate upon multidimensional chromosomes, yielding results substantially better than those previously published.

G5.1.1 Overview: the movement of airborne pollution

Public concern about the health effects of airborne pollution has prompted the installation in industrialized countries of atmospheric pollution monitoring networks (figure G5.1.1), to monitor air quality at strategically placed sampling points. Each node in the network is known as a *receptor*.

The air around receptors is sampled on an intermittent or continuous basis, and the concentration of key pollutants determined at the stations themselves, or after transport to an analytical laboratory for processing. Pollutants of particular interest include NO_x, airborne particulates, arsenic, lead, oxides of sulfur, and volatile organic compounds.

The number of chemicals monitored and the sampling frequency may both be large, so receptors generate a considerable volume of information-rich data, whose interpretation is a challenging task. A superficial analysis is of little value: a report that the concentrations of NO_x rose beyond permitted levels indicates only that a pollution problem existed, not its source. Furthermore, analysis of receptor data must be rapid and reliable, so that if a serious pollution episode occurs, remedial steps can be taken quickly and, if appropriate, legal action against a polluter can be initiated based upon credible scientific evidence.

G5.1.2 The need for an evolutionary solution to source apportionment

The sources of pollution in an urban area may be pseudo-point emitters, such as smelters or power stations, or extended emitters, such as roads, high-density housing, or fireworks (a notable source of pollution at certain times of the year). The pollution released by each source is diluted by the atmosphere, mixed with pollution from other sources, and carried on the prevailing wind before being sampled at receptors. The assessment of data from receptors to determine the source of pollution is the source apportionment problem, which has been investigated for a number of years (Cooper and Watson 1980, Liu *et al* 1982, Currie *et al* 1984, Wang and Hopke 1989). Various approaches have been tried, but in general they have yielded disappointing results.

The raw data available for source apportionment comprise:

(i) the chemical analysis of samples collected at each receptor, and
(ii) the estimated profile of pollutants released by each source.

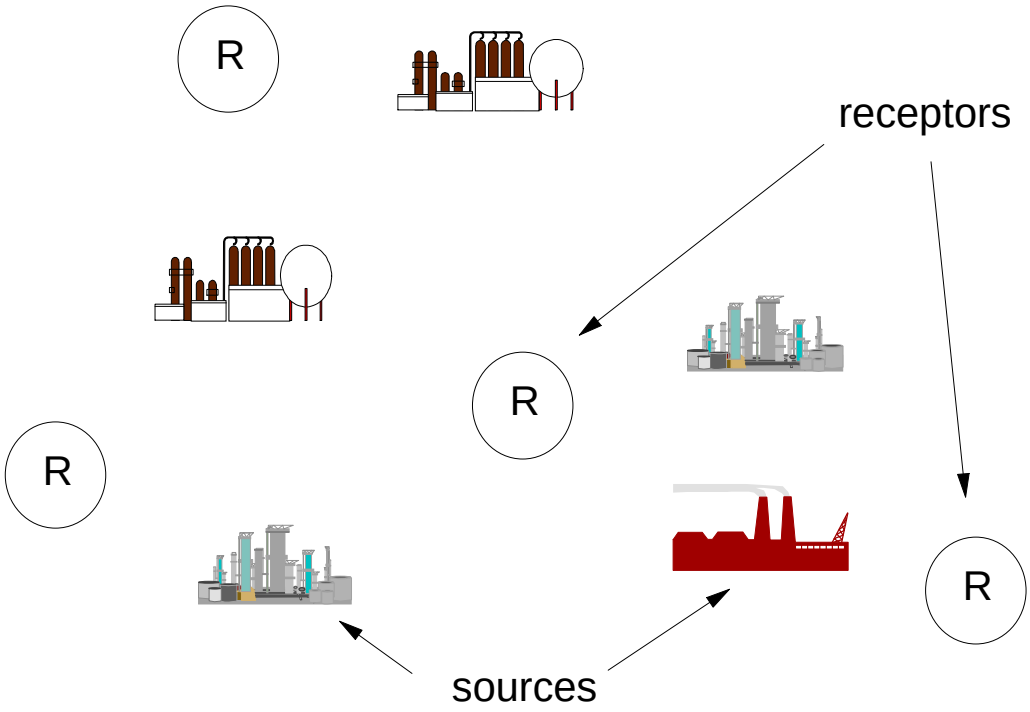


Figure G5.1.1. A network of receptor stations in an urban environment.

These latter data—specifying the identity and quantity of pollution generated at each source—are usually *fuzzy* (that is, they are at best only loosely quantitative). For example, we may know that a smelter emits particulate zinc and cadmium in a ratio which is roughly constant, and determined by the type of ore it processes, but the absolute amounts of metal released may vary substantially and unpredictably during smelting. In the most favorable circumstances, reliable profile data may be available through on-site monitoring of factory smokestack emissions. At the other extreme, data may be so fuzzy as to be almost valueless: large quantities of pollutant may be released accidentally or covertly, without any indication of such release being available to the analysis algorithm. It is clear, then, that considerable uncertainty may exist in the emission data, and that even the size of the uncertainty itself may be difficult to gauge.

The complexity of the problem, and the multidimensional nature of the data, suggest that the *genetic algorithm* (GA) could be a promising method of attack. This expectation is borne out in practice, though the standard GA requires some modification before it can meaningfully outperform conventional methods. B1.2

G5.1.3 Genetic algorithm implementation

G5.1.3.1 Representation

The success of a GA is heavily dependent upon the coding used to represent the problem, and on the form of the fitness function. In early work, GA chromosomes generally were represented in *binary format* (Holland 1975). By contrast, in many applications in science it is convenient to use floating-point values. The internal representation in digital computers, is, of course, binary whatever the coding, but an explicit floating-point representation in the high-level language often simplifies coding and leads to shorter run times. Floating-point strings have been used in this work. C1.2

In a source apportionment calculation the results from chemical analysis of air samples are combined with the fuzzy release profiles; from these data the amount of each pollutant emitted by all significant sources in a geographical region at a given time is determined. It is the emission profiles which are required, so it is apparent that, if we are to use the GA, these profiles must form the GA chromosome. The emission data corresponding to a trial solution might be bolted together to form a GA chromosome in the manner shown in figure G5.1.2, in which the emission levels for all pollutants generated by one source are listed in order, followed by the emission levels for the second source, and so on. However, one can

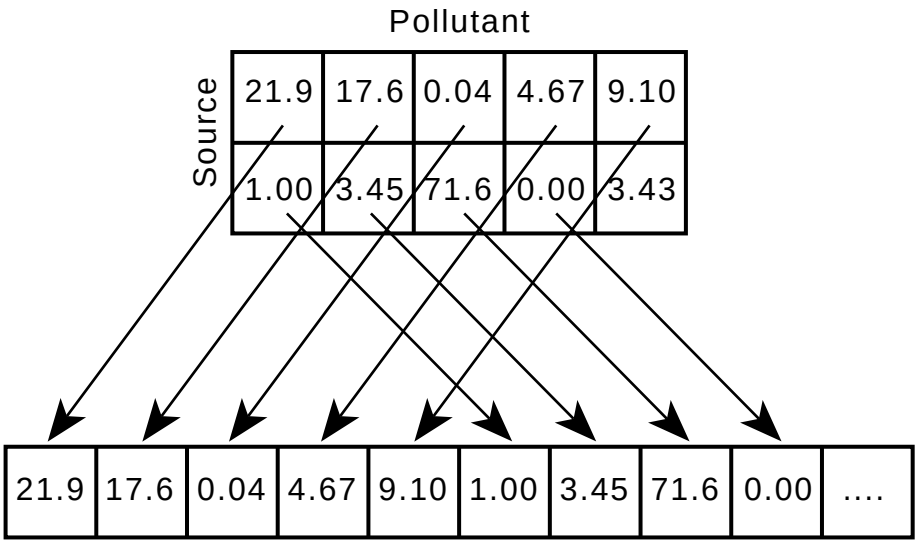


Figure G5.1.2. A possible linear encoding of source emission data.

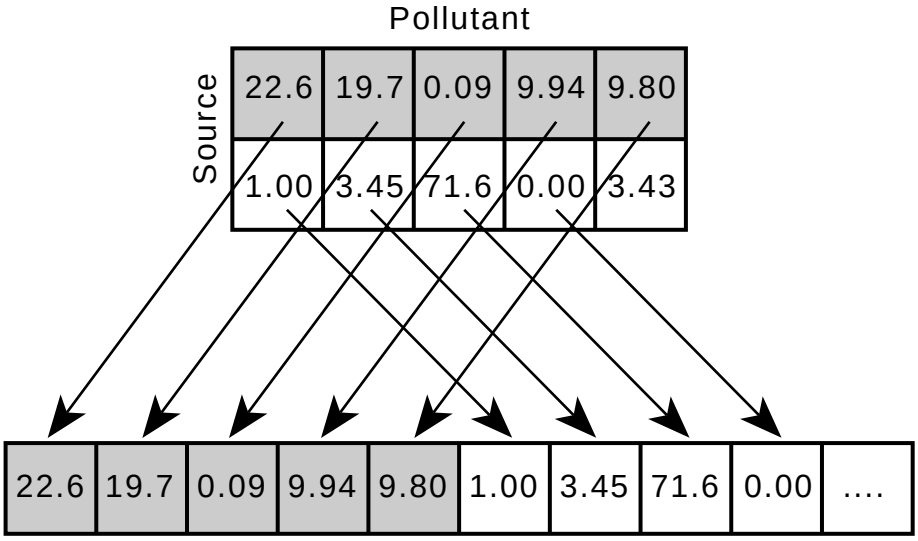


Figure G5.1.3. A linearly encoded string in which a promising solution has been found for a single source (shown shaded).

quickly appreciate that grave difficulties await this type of representation. To understand the nature of these difficulties let us take the (slightly simplistic) view that the GA has as its purpose the shuffling of building blocks (schemata) to build up a solution. It is evident from figure G5.1.2 that positions in the chromosome which represent emissions from a single source are contiguous. Suppose that the algorithm develops a promising solution (shown shaded in figure G5.1.3) for the emissions from one of these sources. When crossover is applied during processing the high-quality information shown shaded is likely to be carried intact from the parent to a child, since it is contained within a short section of the chromosome; destruction through crossover is unlikely because the relevant schema is of short defining length. High-quality source information can therefore apparently be transmitted effectively from one generation to the next.

By contrast, the positions which specify emission of a particular *pollutant* from different sources are spread throughout the chromosome (figure G5.1.4). This wide separation of logically related elements fatally compromises the operation of the GA on linear strings.

If the algorithm were to come across a good solution for a particular pollutant, crossover would almost certainly break up the schema to which this solution corresponds, since the chromosome elements which

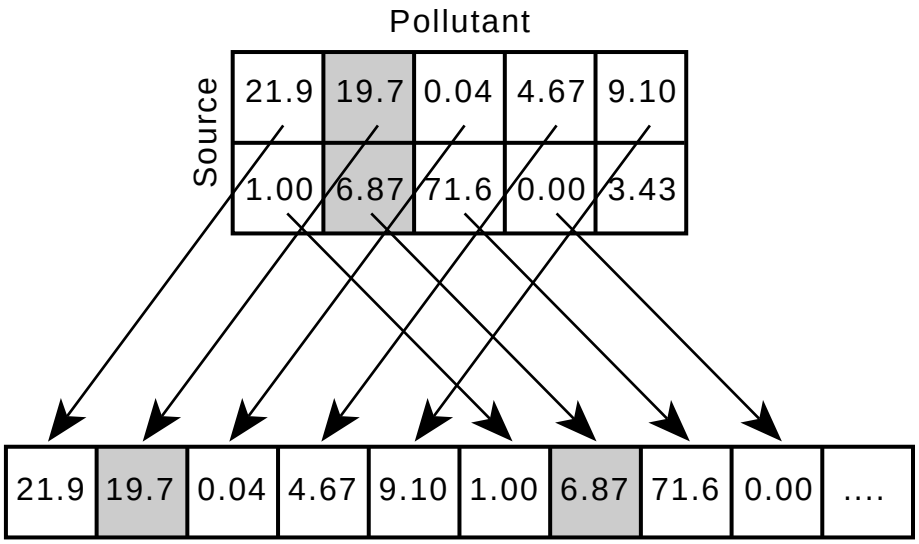


Figure G5.1.4. A linearly encoded string in which a promising solution has been found for a single pollutant (shown shaded).

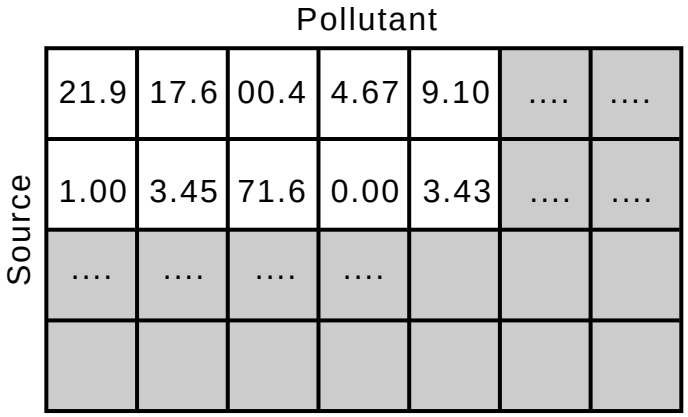


Figure G5.1.5. A two-dimensional source–pollutant GA chromosome.

comprise it are so widely scattered. No linear representation is able to cluster both related source and related pollutant information, so it is unrealistic to expect a GA manipulating linear representations to find solutions which satisfy the twin demands of accurately reproducing emission by source and by pollutant.

A GA chromosome in which the data are arranged in a two- dimensional array need not suffer from this difficulty (figure G5.1.5).

The emission data are naturally cast in matrix form, and, as figure G5.1.5 shows, if we represent these as a two-dimensional chromosome, all values relating to a single source can be located in one row of the chromosome, and all data relating to a particular pollutant in a single column. Thus related data are not dispersed as they are in a linear string. Provided a suitable crossover operator can be constructed, which causes minimal disruption to rows and columns, such a representation should circumvent the difficulty that prevents optimization of one-dimensional chromosomes.

G5.1.3.2 The fitness function

The fitness of a chromosome in the source apportionment GA is determined through calculation of the quantity of pollution of each type that the chromosome *predicts* will arrive at each receptor. The difference between the amount of pollution actually arriving and that predicted is an indication of the quality of the

chromosome, so this is a parameter estimation problem.

$$\frac{1}{f} = \sum_{e=1}^{e_{\max}} \sum_{r=1}^{r_{\max}} \sum_{s=1}^{s_{\max}} |(\text{predicted}[r][s] - \text{experimental}[r][s])|.$$

The summations are over every source, receptor, and pollutant. Summation of (the absolute values of) these differences provides a value whose inverse can be used as a chromosome fitness, but this is too crude a measure to allow the algorithm to converge reliably to acceptable solutions. There are several reasons for this, and these exemplify difficulties which arise (in one form or another) in many scientific applications of the GA.

Firstly, if the absolute values of the differences between predicted and experimental values are summed, the algorithm cannot distinguish between one chromosome in which all pollutants are fitted with reasonable fidelity and another in which nearly all are fitted very well, while a few are fitted poorly. The latter chromosome might be potentially more valuable in the calculation, since one or two mutations or crossover operations might transform it into a chromosome of very high quality. Whatever our preconceptions (or guesses) about which type is more valuable, it might be helpful to the algorithm to have some means of distinguishing between them. To help discriminate, we can form the product of the emission differences, rather than their sum.

$$\frac{1}{f} = \prod_{e=1}^{e_{\max}} \left(\sum_{r=1}^{r_{\max}} \sum_{s=1}^{s_{\max}} |(\text{predicted}[r][s] - \text{experimental}[r][s])| \right).$$

Two further factors which bear upon the fitness of a chromosome relate to scaling. The amounts of different types of pollutant reported by a sampling station may differ by several orders of magnitude, depending upon the type of pollutant, and the units in which the pollutant is measured. Particulate quantities might be quoted as tens or hundreds of particles per cubic metre, while concentrations of airborne arsenic might be 10^{-6} g per cubic metre. If figures such as these were used directly and uncritically, the algorithm would concentrate on fitting the large values and ignore the rest. This would be unhelpful for two reasons. First, the algorithm would be unable to respond if the amounts of a minor pollutant showed an unusually large change. Such a change might be as significant environmentally as a much larger change in a more abundant pollutant, so the algorithm should not ignore it. Just as importantly, by disregarding minor components, the algorithm would fail to take advantage of the information contained in the concentrations of these components (and this information is potentially as valuable as data for those components present at higher levels). Thus a preliminary scaling of the data is desirable to ensure that a similar weight is given to data on all pollutants. (If it is known that certain data are more reliable or more crucial to the analysis than other data, this can of course be taken into account in the weights used in scaling.)

A further valuable modification to the fitness function is the introduction of a power scaling operator. The role of power scaling (with a power < 1.0) is to help prevent premature convergence. It compresses the range of fitnesses, which reduces evolutionary pressure on the less fit. This encourages population diversity, at the expense of a marginal increase in convergence time. Since the population remains more diverse, power scaling reduces the chance that the algorithm will become trapped in a local optimum, and generally produces solutions of higher quality than are obtained if power scaling is not used. The choice of factor for power scaling is made empirically, and in this application a value of 0.6 represents a good compromise between the competing demands of time to convergence and quality of solution.

G5.1.3.3 The crossover operator

The crossover operator processing array chromosomes must swap blocks of the chromosome, rather than linear segments (figure G5.1.6). It is clear that, as a result of treatment by such an operator, the genetic information in a two-dimensional chromosome will suffer less damage than would be suffered by a linear representation.

However, the statement that the crossover operator 'swaps blocks' is loose, and this operation must be considered further. GA operators should treat each position in the chromosome equally, unless there is clear evidence that this is undesirable. For example, if simple two-point crossover is applied to a linear string, the ends of the string are swapped by the operator less frequently than the middle. This difficulty can be overcome using a two-point operator, which wraps around the ends of the string if the second crossover point precedes the first. It is trivial to show that, using this operator, every position has an equal chance of participating in crossover.

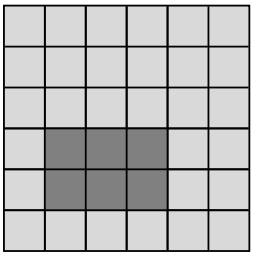


Figure G5.1.6. A two-dimensional block selected for crossover.

In two dimensions, an analogous technique is necessary to ensure that no position is especially favored (Cartwright and Harris 1993). Two-dimensional wrap-around treats the chromosome as a torus, and one, two, or four blocks are swapped between paired strings, depending upon the (x, y) coordinates of the two crossing points (figure G5.1.7).

Similar considerations apply to crossover in three dimensions (Jesson 1995), in which between one and eight blocks may be swapped during crossover (figure G5.1.8).

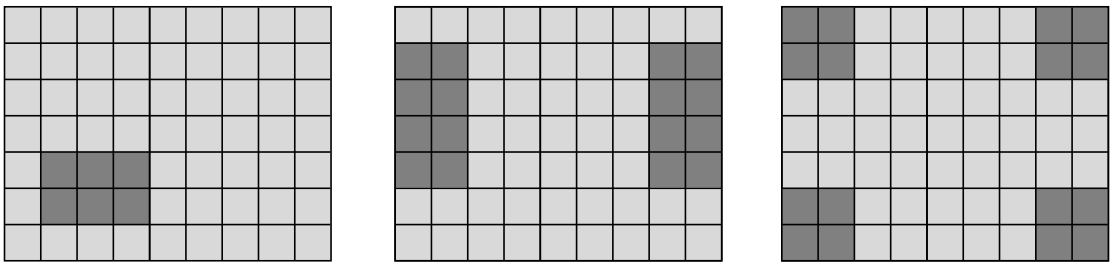


Figure G5.1.7. Wrap-around crossover in two dimensions.

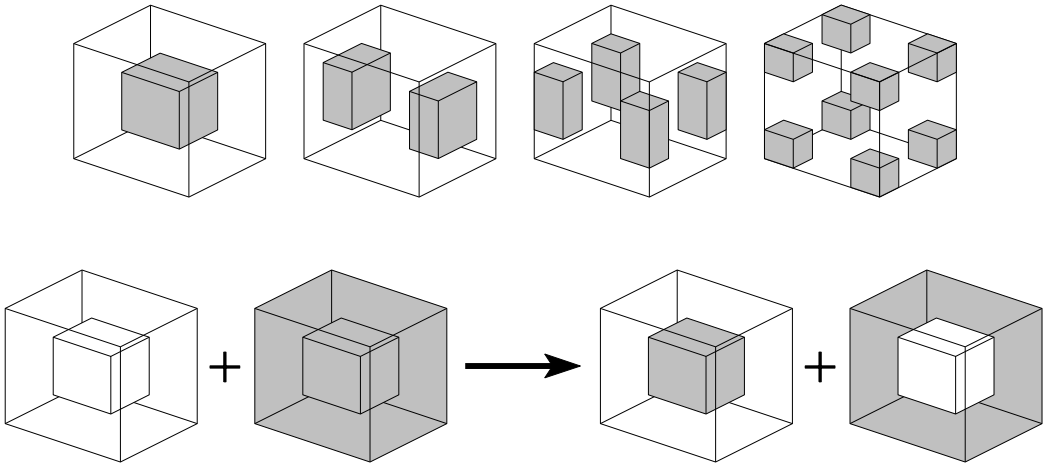


Figure G5.1.8. Wrap-around crossover in three dimensions.

G5.1.3.4 Mutation and local search

As is often the case when floating-point chromosomes are used, the quality of the results in this application can be enhanced by a local search. Within the GA, mutation introduces new numerical data into chromosomes. Mutation is effective when binary-valued genes are used, and clearly can generate all

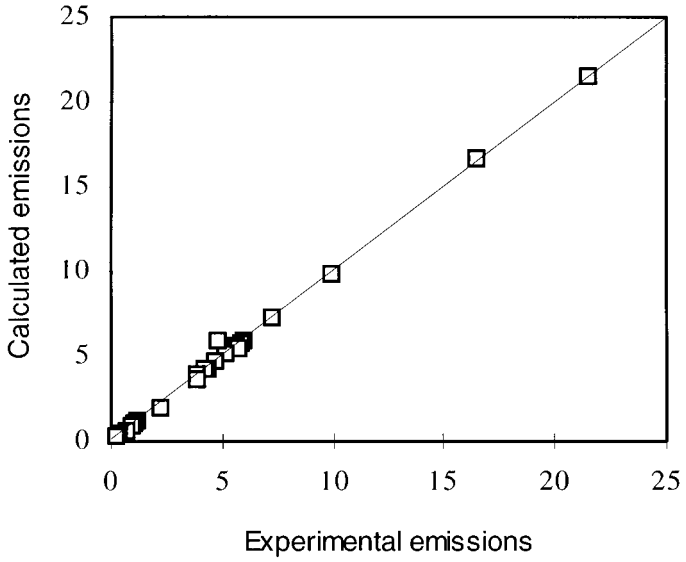


Figure G5.1.9. The correlation between calculated and experimental values for a set of emission data.

possible chromosomes in the search space. When floating point values are used however, so that in principle any whole number may be generated, mutation is less effective. The range of permissible gene values is for practical purposes infinite, and the probability that mutation on its own will generate a particular value within this infinite range is effectively zero. This suggests the algorithm will never completely converge if floating point chromosomes are used. It is common in floating-point applications therefore to use a local search (Reeves and Hohn 1995) which investigates a gradually decreasing range around each value in a chromosome, and we have done so in this application. As local search is comparatively time intensive, it is switched on only when the GA has located promising solutions, and the rate of convergence has begun to diminish.

G5.1.4 Implementation

The model used in these calculations and typical results have been described in detail elsewhere (Cartwright and Harris 1993). Typical parameters used were a population size of 40, mutation probability of 0.05 per chromosome per generation, and crossover probability of 0.7 per chromosome per generation. 'Experimental' pollution data are generated by an environmental model, and the GA then works back from these simulated receptor sets to try to recover the original emission data. Since the calculation makes use of a model set of data, the quality of the solutions found by the GA can be assessed by plotting a scatter graph of calculated against experimental data. A typical set of results is shown in figure G5.1.9.

It can be seen that there is good agreement between calculated and experimental data, with an average deviation of around 4%. This compares favorably with similar work in the literature using conventional methods (Liu *et al* 1982) in which the average deviations were approximately 90%. A good correlation between experimental and predicted values has also been found when the environmental data were overlaid with random Gaussian noise, to simulate real data.

G5.1.5 Discussion

This application of GAs to source apportionment illustrates several points of particular relevance to the analysis of scientific data.

Perhaps most crucially, it emphasizes the importance of choosing a suitable format in which to code the problem—a central concern in the planning of a GA calculation and one which has been broadly discussed in the literature. Preliminary trials confirmed that the GA would be unable to develop acceptable solutions using linear chromosomes. Similar conclusions have been drawn for a second environmental problem—the analysis of waste flow from multiunit chemical complexes—in which it has been shown that

three-dimensional chromosomes are required if the algorithm is to locate high-quality solutions (Jesson 1995).

Secondly, scaling of scientific data is often required before analysis, since values of the parameters which define the problem, such as concentrations of different components or the times at which different types of event occur, are often of quite different magnitudes. Without scaling, the GA can be expected to focus on the parameters of largest magnitude, ignoring minor components. This has clearly demonstrable negative effects upon the quality of fit.

Thirdly, although binary coding is widely used, floating-point coding is often conceptually simple and fast. The relative merits of binary and floating point coding have been discussed with enthusiasm in the GA community, but there is little doubt that, in many scientific applications, floating point coding is both concise and effective.

A drawback of floating-point coding is the increase in search space, so *local search* techniques are valuable. However, these increase the danger that, at an early stage in the calculation, a solution may be found which is so superior to all others in the population that the algorithm converges prematurely on a local optimum. This danger may be reduced by employing local search with a light touch early on, so that initially it is no more intrusive than mutation; the depth of the local search can then gradually be increased. Alternatively, local search might not be used at all until the GA has discovered promising solutions, and the rate of convergence has subsided to a fairly low level. D3.2

Scientific problems are often rather different in nature from the discrete scheduling and routing applications which have come to be regarded as a particular strength of GAs. It is evident from the increasing number of papers discussing the application of GAs to science that, provided the special characteristics of each problem are allowed for, GAs form a promising tool in the analysis of scientific data.

References

Cartwright H M and Harris S P 1993 Analysis of the distribution of airborne pollution using genetic algorithms *Atmos. Environ.* A **27** 1783–91

Cooper J A and Watson J G 1980 Receptor oriented methods of air particulate source apportionment *J. Air Pollut. Control Ass.* **30** 1116–25

Currie L A *et al* 1984 Interlaboratory comparison of source apportionment procedures: results for simulated sets *Atmos. Environ.* **18** 1517–37

Harris S P 1991 *Chemical Mass Balance Calculations using Genetic Algorithms* Chemistry Part II Thesis, Oxford University

Holland J 1975 *Adaptation in Natural and Artificial Systems* (Ann Arbour, MI: University of Michigan Press)

Jesson B J 1995 *Chemical Waste Flow Analysis using Genetic Algorithms* Chemistry Part II Thesis, Oxford University

Liu C-K *et al* 1982 The application of factor analysis to source apportionment of aerosol mass *Am. Ind. Hyg. Assoc. J.* **43** 314–8

Reeves C and Hohn C 1995 Integrating local search into genetic algorithms *Proc. Applied Decision Technol. Conf. (Brunel University, London)* ed V J Rayward-Smith (Uxbridge: Unicom Seminars) pp 261–76

Wang D and Hopke P K 1989 The use of constrained least-squares to solve the chemical mass balance problem *Atmos. Environ.* **23** 2143–50

G6.1 Classifying protein segments as transmembrane domains using genetic programming and architecture-altering operations

John R Koza

Abstract

This case study describes how the biological theory of gene duplication described in Susumu Ohno's provocative book, *Evolution by Means of Gene Duplication*, was brought to bear on a vexatious problem from the domain of automated machine learning in the computer science field. The resulting biologically motivated approach using six new architecture-altering operations enables genetic programming to automatically discover the size and shape of the solution at the same time as it is evolving a solution to the problem. Genetic programming with the architecture-altering operations was used to evolve a computer program to classify a given protein segment as being a transmembrane domain or nontransmembrane area of the protein (without biochemical knowledge, such as hydrophobicity values). The best genetically evolved program achieved an out-of-sample error rate that was better than that reported for other previously reported human-constructed algorithms. This is an instance of an automated machine learning algorithm that is competitive with human performance on a nontrivial problem.

G6.1.1 Background on genetic programming

The goal of automatic programming is to create, in an automated way, a computer program that enables a computer to solve a problem. Ideally, an automatic programming system should require that the user prespecify as little as possible about the problem. In particular, it is desirable that the user not be required to specify the size and shape (i.e. the architecture) of the ultimate solution to the problem before applying the technique. One of the banes of automated machine learning from the earliest times has been the requirement that the human user predetermine the size and shape of the ultimate solution to his problem (Samuel 1959). I believe that the size and shape of the solution should be part of the *answer* provided by an automated machine learning technique, rather than part of the *question* supplied by the investigator.

John Holland's pioneering *Adaptation in Natural and Artificial Systems* (Holland 1975) described how an analog of the naturally occurring evolutionary process can be applied to solving problems using what is now called the *genetic algorithm*. B1.2

The book *Genetic Programming: On the Programming of Computers by Means of Natural Selection* (Koza 1992) describes an extension of the genetic algorithm in which the genetic population consists of computer programs, that is, compositions of primitive functions, terminals, and possibly automatically defined functions (see Section B1.5 of this handbook). In a run of genetic programming in its most basic form, the size and shape of the result-producing program as well as the sequence of work-performing steps are evolved. A videotape description of genetic programming can be found in the book by Koza and Rice (1992). Recent research activity in genetic programming is described by Kinnear (1994), Angeline and Kinnear (1996), and Koza and coworkers (1996). B1.5

I believe that no approach to automated programming is likely to be successful on nontrivial problems unless it provides some hierarchical mechanism to exploit, *by reuse* and parametrization, the regularities,

symmetries, homogeneities, similarities, patterns, and modularities inherent in problem environments. Subroutines do this in ordinary computer programs. Accordingly, *Genetic Programming II: Automatic Discovery of Reusable Programs* (Koza 1994a) describes how to evolve multipart programs consisting of a main program and one or more reusable, parametrized, hierarchically called subprograms. An *automatically defined function* is a function (i.e. subroutine, procedure, DEFUN module) that is dynamically evolved during a run of genetic programming in association with a particular individual program in the population and which may be invoked by a calling program (e.g. a main program) that is simultaneously being evolved. A description of automatically defined functions can be found in the videotape by Koza (1994b).

When automatically defined functions are being evolved in a run of genetic programming, it becomes necessary to determine the architecture of the overall program to be evolved. The specification of the architecture consists of (i) the number of function-defining branches (automatically defined functions) in the overall program, (ii) the number of arguments (if any) possessed by each function-defining branch, and (iii) if there is more than one function-defining branch, the nature of the hierarchical references (if any) allowed between the function-defining branches.

The question of how to specify the architecture of the overall program in genetic programming has a parallel in the biological world: how are new structures and behaviors created in living things? This corresponds to the question of how new proteins are created in more complex organisms.

In nature, recombination ordinarily recombines a part of the chromosome of one parent with a corresponding (homologous) part of the second parent's chromosome. A gene duplication is a rare illegitimate recombination event that results in the duplication of a possibly lengthy subsequence of a chromosome. Susumu Ohno's seminal book *Evolution by Gene Duplication* (Ohno 1970) proposed the then-provocative (now accepted) thesis that the creation of new proteins (and hence new structures and behaviors in living things) begins with a gene duplication and that gene duplication is 'the major force of evolution'. Ohno claimed that simple point mutation and crossover are insufficient to explain major evolutionary changes:

> ...while allelic changes at already existing gene loci suffice for racial differentiation within species as well as for adaptive radiation from an immediate ancestor, they cannot account for large changes in evolution, because large changes are made possible by the acquisition of new gene loci with previously nonexistent functions.

The naturally occurring mechanism of gene duplication (and the complementary mechanism of gene deletion) motivated the addition of six new architecture-altering operations to genetic programming (Koza 1994d, 1995). These operations of branch duplication, branch creation, branch deletion, argument duplication, argument creation, and argument deletion enable genetic programming to evolve the architecture of a multipart program containing automatically defined functions (ADFs) during a run of genetic programming. The operations enable the analog of what Ohno described as 'the acquisition of new gene loci with previously nonexistent functions'.

G6.1.2 Classifying protein segments as transmembrane domains

This paper considers the problem of deciding whether a given protein segment is a transmembrane domain or nontransmembrane area of the protein.

Proteins are responsible for such a wide variety of biological structures and functions that it can be said that the structure and functions of living organisms are primarily determined by proteins (Stryer 1995). Proteins are polypeptide molecules composed of sequences of amino acids. There are 20 amino acids (also called residues) in the alphabet of proteins (denoted by the letters `A`, `C`, `D`, `E`, `F`, `G`, `H`, `I`, `K`, `L`, `M`, `N`, `P`, `Q`, `R`, `S`, `T`, `V`, `W`, and `Y`). Automated methods of machine learning may prove to be useful in discovering biologically meaningful information hidden in the rapidly growing databases of DNA sequences and protein sequences.

Membranes play many important roles in living things. A *transmembrane protein* (Yeagle 1993) is embedded in a membrane in such a way that part of the protein is located on one side of the membrane, part is within the membrane, and part is on the opposite side of the membrane. Transmembrane proteins often cross back and forth through the membrane several times and have short loops immersed in the different milieux on each side of the membrane. Understanding the behavior of transmembrane proteins requires identification of the portion(s) of the protein that are actually embedded within the membrane, such portion(s) being called the *transmembrane domain(s)* of the protein. The lengths of the transmembrane

domains of a protein are usually different from one another and the lengths of the nontransmembrane domains are also usually different from one another.

Algorithms written by biologists for the problem of classifying transmembrane domains in protein sequences are based on biochemical knowledge about hydrophobicity and other properties of membrane-spanning areas of the protein sequence (Kyte and Doolittle 1982, von Heijne 1992, Engelman *et al* 1986).

This problem provides an opportunity to illustrate automatic discovery of reusable feature detectors, the evolution of the architecture of a multipart computer program using the architecture-altering operations, the use of state (memory), and the use of iteration-performing steps (in conjunction with information stored in memory) in genetically evolved computer programs. In this section, genetic programming will be given a set of differently sized protein segments and asked to give the correct classification for each segment.

Genetic programming has previously demonstrated the ability to evolve a classifying program for this task without using any biochemical knowledge (Koza 1994c) when the user specified the architecture of the program to be evolved. The genetically evolved program achieved a better error rate than the three human-written algorithms that were compared as well as the algorithm developed by Weiss *et al* (1993) using human knowledge along with an element of machine learning.

We now solve this problem again using the architecture-altering operations. The goal is to find a classifying program consisting of an *initially unspecified* number of automatically defined functions, each function possessing an *initially unspecified* number of arguments, consisting of an *initially unspecified* sequence of work-performing operations, an initially *unspecified* sequence of work-performing operations in an iterative calculation, and an initially *unspecified* final result-producing calculation that yields a classification of the protein segment.

The function set for each branch of each program to be evolved consists of four arithmetic operations: (i) a three-argument conditional branching operator, (ii) a one-argument setting function, SETM0, that sets the settable memory variable, M0, to a particular value, and (iii) a two-argument numerical-valued disjunctive function.

The terminal set consists of the settable variable, M0, floating-point random constants, the length of the protein segment being examined, and 20 zero-argument amino-acid-detecting functions that enable the program to examine the protein segment.

Fitness is the correlation between the classification produced by an evolved program and the correct classification. An in-sample (training) set of protein segments is used during the evolutionary process; an out-of-sample (testing) set is used to measure and report the performance of the best program produced by a run.

The population size was 128 000. The problem (written in ANSI C) was run on a medium-grained parallel Parsytec computer system consisting of 64 Power PC 601 processors arranged in a toroidal mesh with a host PC Pentium-type computer (running *Windows*). The Power PC processors communicated by means of one INMOS transputer that was associated with each Power PC processor. The so-called *distributed genetic algorithm* or *island model* for parallelization was used (Goldberg l989). That is, subpopulations (called *demes* after Wright (1943)) were situated at the processing nodes of the parallel system. The population size was $Q = 2000$ at each of the $D = 64$ demes for a total population size of 128 000. The initial random subpopulations were created locally at each processing node. Generations were run asynchronously on each node. After a generation of genetic operations was performed locally on a given node, four boatloads, each consisting of $B = 5\%$ (the migration rate) of the subpopulation (selected on the basis of fitness), were dispatched to each of the four toroidally adjacent nodes. Details of this parallel implementation of genetic programming (and a comparative discussion of migration rates) can be found in the articles by Koza and Andre (1995) and Andre and Koza (1996). C6.3

On the first run (23 hours) with genetic programming and the architecture-altering operations, a solution was obtained for this problem that exceeded the performance of the three human-written algorithms as well as the algorithm developed by Weiss *et al* (1993). The best program of generation 28 scores an in-sample correlation of 0.9596, an out-of-sample correlation of 0.9681, an in-sample error rate of 3%, and an out-of-sample error rate of 1.6%. There were 246 fitness cases (half negative and half positive) in the in-sample set of fitness cases, and there were 250 fitness cases (again, half negative and half positive) in the out-of-sample set of fitness cases (as described in detail in chapter 18 of Koza (1994a)).

Figure G6.1.1 shows the high-level architecture of the best-of-run program from generation 28. This program has one automatically defined function, ADF0, that tests for the amino acid residues phenylalanine (F) and leucine (L), one 36-point iteration-performing branch, IPB, and one 169-point result-producing branch, RPB.

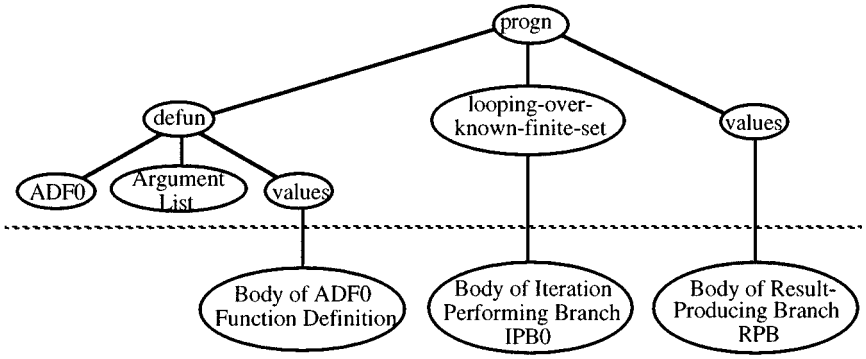


Figure G6.1.1. High-level architecture of best-of-run program from generation 28 with one zero-argument automatically defined function, ADF0, that tests for certain amino acid residues in the protein segment, one 36-point iteration-performing branch, IPB0, and one 169-point result-producing branch, RPB.

After genetic programming evolves a solution to a problem, it is often difficult to analyze the program produced by the evolutionary process. However, a number of fortuitous circumstances permitted this particular evolved program to be simplified, by hand, to the following procedure:

(i) Create a sum, S, by adding four for each `E` in the protein segment and two for each `C, D, G, H, K, N, P, Q, R, S, T, W`, or `Y` (i.e. the 13 residues that are neither `E` nor `A, M, V, I, F`, or `L`) in the protein segment.

(ii) If

$$\left[\frac{S - 3.1544}{0.9357}\right] < \text{LEN}$$

where LEN is the length of the protein segment, then classify the protein segment as a transmembrane domain; otherwise, classify it as a nontransmembrane area of the protein.

This genetically evolved procedure is simple and works because of the high hydrophobicity of the six amino acid residues `A, M, V, I, F`, and `L`.

Table G6.1.1 shows the out-of-sample error rate for the four previous algorithms for classifying transmembrane domains as well as for three approaches using genetic programming, namely the set-creating version (sections 18.5 through 18.9 of Koza 1994a), the arithmetic-performing version (sections 18.10 and 18.11 of Koza 1994a), and the version using the architecture-altering operations as reported herein.

Table G6.1.1. A comparison of seven methods.

Method	Error rate (%)
von Heijne (1992)	2.8
Engelman, Steitz and Goldman (1986)	2.7
Kyte and Doolittle (1982)	2.5
Weiss, Cohen and Indurkhya (1993)	2.5
GP + set-creating ADFs of Koza (1994a)	1.6
GP + arithmetic-performing ADFs of Koza (1994a)	1.6
GP + ADFs + architecture-altering operations (this paper)	1.6

G6.1.3 Conclusion

We have shown that it is possible to evolve the architecture of a multipart program, while concurrently solving the problem, for the problem of classifying protein segments as transmembrane domains or nontransmembrane areas of the protein.

The architecture-altering operations executed during the run of genetic programming determined the existence and eventual number of the automatically defined functions, the number of arguments possessed

by each automatically defined function, the size, shape, and sequence of work-performing steps within the automatically defined functions, the size, shape, and sequence of work-performing steps in the iteration-performing branch, and the size, shape, and sequence of work-performing steps in the result-producing branch.

The solution to the problem of classifying transmembrane domains in protein segments is slightly better than the performance of algorithms written by knowledgeable human investigators. This is an instance of an automated machine learning algorithm slightly exceeding human performance on a nontrivial problem.

Acknowledgments

David Andre and Walter Alden Tackett wrote the computer program in ANSI C to implement five of the architecture-altering operations described above.

References

Andre D and Koza J R 1996 Parallel genetic programming: a scalable implementation using the transputer network architecture *Advances in Genetic Programming 2* ed P J Angeline and K E Kinnear Jr (Cambridge, MA: MIT Press) ch 18

Angeline P J and Kinnear K E Jr (eds) 1996 *Advances in Genetic Programming* (Cambridge, MA: MIT Press)

Engelman D, Steitz T and Goldman A 1986 Identifying nonpolar transbilayer helices in amino acid sequences of membrane proteins *Ann. Rev. Biophys. Biophysiol. Chem.* **15**

Goldberg D E 1989 *Genetic Algorithms in Search, Optimization and Machine Learning* (Reading, MA: Addison-Wesley)

Holland J H 1975 *Adaptation in Natural and Artificial Systems: an Introductory Analysis with Applications to Biology, Control and Artificial Intelligence* (Ann Arbor, MI: University of Michigan Press) (1992 2nd edn Cambridge, MA: MIT Press)

Kinnear K E Jr (ed) 1994 *Advances in Genetic Programming* (Cambridge, MA: MIT Press)

Koza J R 1992 *Genetic Programming: On the Programming of Computers by Means of Natural Selection* (Cambridge, MA: MIT Press)

——1994a *Genetic Programming II: Automatic Discovery of Reusable Programs* (Cambridge, MA: MIT Press)

——1994b *Genetic Programming II: Videotape: The Next Generation* (Cambridge, MA: MIT Press)

——1994c Evolution of a computer program for classifying protein segments as transmembrane domains using genetic programming *Proc. 2nd Int. Conf. on Intelligent Systems for Molecular Biology* ed R Altman, D Brutlag, P Karp, R Lathrop and D Searls (Menlo Park, CA: AAAI) pp 244–52

——1994d *Architecture-Altering Operations for Evolving the Architecture of a Multi-Part Program in Genetic Programming* Technical Report STAN-CS-TR-94-1528, Computer Science Department, Stanford University

——1995 Gene duplication to enable genetic programming to concurrently evolve both the architecture and work-performing steps of a computer program *Proc. 14th Int. Joint Conf. on Artificial Intelligence* (San Francisco, CA: Morgan Kaufmann) pp 734–40

Koza J R and Andre D 1995 *Parallel Genetic Programming on a Network of Transputers* Technical Report STAN-CS-TR-95-1542, Computer Science Department, Stanford University

Koza J R, Goldberg D E, Fogel D B and Riolo R L (ed) 1996 *Genetic Programming 1996: Proc. 1st Ann. Conf. on Genetic Programming (Stanford, 1996)* (Cambridge, MA: MIT Press)

Koza J R and Rice J P 1992 *Genetic Programming: The Movie* (Cambridge, MA: MIT Press)

Kyte J and Doolittle R 1982 A simple method for displaying the hydropathic character of proteins *J. Mol. Biol.* **157** 105–32

Ohno S 1970 *Evolution by Gene Duplication* (New York: Springer)

Samuel A L 1959 Some studies in machine learning using the game of checkers *IBM J. Res. Dev.* **3** 210–29

Stryer L 1995 *Biochemistry* 4th edn (New York: Freeman)

von Heijne G 1992 Membrane protein structure prediction: hydrophobicity analysis and the positive-inside rule *J. Mol. Biol.* **225** 487–94

Weiss S M, Cohen D M and Indurkhya N 1993 Transmembrane segment prediction from protein sequence data *Proc. 1st Int. Conf. on Intelligent Systems for Molecular Biology* ed L Hunter, D Searls and J Shavlik (Menlo Park, CA: AAAI Press)

Wright S 1943 Isolation by distance *Genetics* **28** 114–38

Yeagle P L 1993 *The Membranes of Cells* 2nd edn (San Diego, CA: Academic)

G7.1 Modeling economic interaction using a genetic algorithm

Edmund Chattoe

Abstract

This case study describes an application of the genetic algorithm to the modeling of an economic process, the interaction of competing firms in a market. It distinguishes instrumental applications of evolutionary algorithms, which are designed to perform a given task as quickly and efficiently as possible, from descriptive applications, which are intended to enhance our understanding of the process which they describe. It argues that, not surprisingly, descriptive and instrumental applications of evolutionary computation have different perspectives and requirements for success. It illustrates these differences by describing some difficulties that arise in modeling economic interaction using an evolutionary algorithm. It also suggests some distinctions that may be useful in avoiding these difficulties. The article uses the work of Arifovic on modeling the convergence of interacting firms to rational expectations equilibrium as a basis for discussion.

G7.1.1 Introduction

A distinction can be made between instrumental and descriptive applications of evolutionary computation. Instrumental applications perform tasks, such as face recognition or bin packing, largely generated in commercial or practical spheres outside the academic community. Their development is chiefly motivated by the need to carry out these tasks quickly and accurately. Any contribution to the theory of evolutionary computation is of secondary importance. Ideally an instrumental application should also demonstrate robustness and the ability to generalize well over other problems in as wide a class as possible. By contrast, a descriptive use of evolutionary computation occurs when a physical or social system appears to exhibit similar behavior to an evolutionary algorithm. Here the criteria for successful development are more complicated. The understanding of both the algorithm and the process it models must be extended until it is possible to specify a plausible and complete interpretation (or analogy) for the operation of the algorithm in terms of the process being modeled. Unlike an instrumental application, a descriptive one should increase our understanding of the process which it describes. It may do this in several ways, for example, by increasing quantitative or qualitative predictive accuracy, or by suggesting relevant issues for further research. (This explains the relative scarcity of instrumental applications in academia. Research places far greater emphasis on description and understanding as goals in their own right.)

Historically, instrumental applications have been dominant in all areas of evolutionary computation, though this preoccupation has been questioned (De Jong 1992). However, a small number of researchers have applied the descriptive approach to social processes, notably in economic theory. There is a long tradition of evolutionary ideas in that discipline but the absence of a formal framework for modeling has marginalized the resulting discussions. (For a detailed description of the history of evolutionary ideas in economics, see the book by Hodgson (1991). The earliest book devoted to evolutionary processes in economics is that by Nelson and Winter, published only in 1982.)

G7.1.2 The challenge of descriptive models

The descriptive use of evolutionary computation reveals a number of tensions that do not arise in purely instrumental applications. Although these tensions do not, on balance, suggest the complete rejection of descriptive evolutionary models of social processes, they do suggest that the use of such models requires considerable caution.

There are three overlapping considerations in the application of evolutionary computation to social processes (Chattoe 1994):

(i) To what extent does the descriptive model provide a satisfactory analogy with a social process? Is the analogy completely specified and can its individual parts be supported independently and empirically? (The latter requirement is important because it distinguishes analogies which are robust enough to support further theorizing or data collection, and thus to be falsified, from those which simply redescribe a phenomenon in new terms.)

(ii) Does the model adequately acknowledge the current development of evolutionary computation, for example in its choice of appropriate genetic operators, or is the model chosen naively, simply to produce some 'appropriate' result? Does the analogy develop both the techniques of social science and evolutionary computation? Since descriptive models using evolutionary computation are part of the appropriate physical or social science rather than part of engineering or computer science, it must be demonstrated that theories of this type have 'heuristic fertility' on both sides of the analogy, that is, they should not only increase our understanding, but suggest directions in which that understanding can be developed further. Although the fertility of an analogy cannot be quantified, qualitative performance is usually apparent in the progress of subsequent research.

(iii) What is the motivation for using a descriptive model based on evolutionary computation in modeling a particular social process? Is the application motivated by empirical or theoretical suitability?

The attempt to meet these challenges is particularly well illustrated by the work of Arifovic (1990, 1994), which has been chosen for discussion here.

G7.1.3 The model

Arifovic (1990, 1994) uses two models of the economic decision-making of individual firms in a market which are based on a *genetic algorithm* (GA). In the first, the GA represents a population of firms. B1.2 In the second, each firm uses a decision process that operates in a similar manner to a GA, where the credence given to each strategy by a firm depends on its success. (I shall concentrate on the first interpretation, since it is developed further in the course of the papers considered here.) Arifovic applies both approaches to a number of economic models of imperfect competition between firms that are already well understood mathematically. Her conclusion is that in a variety of situations the GA can model convergence to the theoretically important rational expectations equilibrium. This is a situation in which each firm's expectation of the market price in a given period is equal to the actual price. Each firm has effectively learned both the correct model of the environment and the actual parameter values of that model. Furthermore, she shows that the GA produces more robust convergence, using less information, than a number of learning algorithms that have also been applied to the same models. The convergence to any particular equilibrium is less sensitive to initial conditions and convergence to some equilibrium can occur from a larger set of initial parameter values. (Less information is involved because the operation of the GA does not require the calculation of gradients, or other derived information, to direct the process of search.) The dynamics of the GA prior to convergence also correspond more closely to the results obtained in experimental studies of the convergence process. Finally, Arifovic demonstrates an important negative result, that rational expectations equilibrium cannot be attained by a traditional GA in the so-called cobweb model which has been thoroughly studied. However, the addition of an 'election operator' allows convergence even in a dynamic environment, for example where the demand function changes gradually over time. This result echoes the work of Rudolph (1994) demonstrating the need for some form of 'elitism' to guarantee convergence in a simple GA.

Arifovic describes the analogy between the market process and the GA in some detail. The individual strings in the GA represent codings of the decision of each firm concerning how much to produce in each period. The fitness is the amount of profit resulting from this decision given both a fixed cost, independent of the quantity produced, and a variable cost, which is not. Reproduction 'works like the imitation of

successful rivals' (1994, p 10). Crossover and mutation 'are used to generate new ideas (beliefs) on how much to produce' (p 11). Using the election operator 'firms generate new production decisions using genetic operators. They compare the fitness of these new potential proposals to the old set, under the market conditions observed in the past. Only new ideas that appear promising on such grounds are actually implemented' (p 11).

As a descriptive model, this interpretation can be considered from the perspective of both economics and evolutionary computation. The functioning of the genetic operators is that of a standard GA, but the fitness function is unusual in that raw fitness depends on the actions of other firms, through their effect on market price. (In many fitness functions, the calculation of *relative* fitness may involve normalization by the fitness of all other individuals, but here the profit function is directly dependent on the actions of others. An excessive production level may result in a negative profit for all firms.) There is a fairly straightforward interpretation of relative success for firms making positive profits, in terms of credibility and retained funds (Alchian 1950), but this interpretation breaks down for firms making negative profits and one supposes that such a market would simply collapse altogether, rather than allowing the fittest firms, those with the smallest losses, to survive for any significant period of time.

Even if these concerns can be addressed by the selection of a more appropriate GA, issues raised by the economic side of the interpretation are rather more challenging. In an instrumental GA, reproduction actually consists of the production of genuinely new individuals (offspring), at least notionally. Arifovic suggests that although firms remain physically the same, they imitate the strategies of more successful firms. In terms of what firms are actually supposed to be doing in this model, reproduction is more like a form of crossover than the generation of new firms with the same properties as the old. That is to say that the new strategy comes from a rival firm by observation, rather than being 'passed on,' for example by instruction. This distinction is important for two reasons. In the first place, such straightforward imitation is only plausible when firms are only capable of extremely simple actions. (It should be noted that the processes of crossover and mutation are actually implausibly complex ways of discovering a new quantity to produce. The interpretation of mixing strategies of such simplicity is rather strained.) If firms consisted of an underlying genotype or adaptive strategy (one in which the observed behavior depended on previous behavior and/or the current state of the world), it would be extremely difficult to imitate this strategy or deduce it purely from such simple actions as quantity setting to which it gave rise. Such a strategy, being adaptive, would also lead to varying actions over time, making interpretation even harder and excluding all imitation except 'blind' follower behavior that would have to be adjusted in each round. Secondly, firms that could be regarded as 'offspring' would be far less likely to suffer these difficulties than those which operated by imitation. (In the competition of fast food chains for example, each new branch can be seen as a success-based 'offspring,' capturing a new sector of the market from its rivals and accurately reproducing the operating procedures of the other branches. Casual observation suggests that outlets that merely imitate the decor or product range of the well known are far less successful!)

These difficulties raise two underlying issues. Firstly, the absence of a significant distinction between genotype and phenotype is more relevant to descriptive models. The realism of the model depends very much on whether we assume that firms can change strategy or are simply *defined* as following a single strategy. In the model described here, firms are effectively no more than their production decisions. This simple view derives from the instrumental use of the GA where there is really nothing to distinguish a parent from its identical offspring: phenotype and genotype are degenerate, because there is no use to their being otherwise. Another manifestation of this degeneracy is the fact that both election and imitation are interpreted as processes originating inside individual firms but they are *modeled* as probabilities that are exogenous and fixed. As a result, it appears that this model simply pushes the task of explanation back one stage. Instead of explaining how firms converge on equilibrium, we now have to explain how it is that they have the correct reproduction and election rates to allow them to reach equilibrium by social evolution. There is no representation of the decision to imitate in the genotype of individual firms. This is unfortunate, because it abstracts from an important feature of biological evolution, that intermediate structures to enhance evolution, such as the ability to reproduce sexually, can themselves be evolved. (Since the speed of adaptation and selection is related to the rate of genetic mixing, sexual reproduction effectively increases adaptiveness. Even though very little genetic novelty is actually beneficial, an increase in the maximum possible rate of mixing is still advantageous.) Intuition suggests that crossover rates that were too high would lead to a market where price cycles and overshooting were observed. If a few firms were doing well, solely because they were using a minority strategy, then they would be imitated by almost all firms in the next period, thus rendering that strategy ineffective. (Recall that in this GA, the strategies

of individual firms are dependent on the strategies of all other firms in the market. Strategies that are good for some firms cannot therefore be assumed to be good for all firms.) By contrast, if operator rates were too low, this would result in markets that did not converge. (Both cycling and nonconvergence are observed in incorrectly tuned GAs used for instrumental applications.) It is therefore important for the plausibility of the model that the 'correct' operator probabilities can be justified within the theory. The process by which profits are observed and used to decide whether or not to imitate is also modeled exogenously and assumed to involve no noise. This is rather surprising as the interpretation of the complex information available to firms constitutes a major part of the uncertainty facing them. To what extent should the profit of a given firm be seen as a function of its output decision, and to what extent a function of circumstances beyond its control? Firms in this model make no attempt to assess the behavior of the market as a whole. As Olivetti (1994) has pointed out, the addition of noise to the model used by Arifovic destroys the convergence result.

Finally, in addition to the fact that the convergence results are not robust to the addition of noise, their value can be questioned on theoretical grounds. The GA is particularly suitable for badly behaved problems where information about differentials or gradients in the search space are not readily calculable and where simple hill climbing algorithms will suffer from premature convergence or cycling. The models to which the GA is applied here are typically well behaved, so the convergence of the GA is hardly surprising. Therefore, except to the extent that the GA is a more realistic description of the social process leading to equilibrium, it is not particularly useful either. Although Arifovic recognizes this point, it does not seem to motivate any investigation of more challenging problems. In these more complex problem spaces, it seems highly likely that the election operator, another endogenous part of the decision process modeled exogenously, would not result in optimality, but considerably impair the efficiency of the GA by premature convergence or nonconvergence. This may explain the results obtained by Olivetti. Unfortunately, such complex problem spaces are not popular in economic theory, precisely because their irregularity renders them unsuitable for solution by the analytic techniques of calculus.

G7.1.4 Analysis

In this section, I shall consider the extent to which the work of Arifovic is able to address the problem of descriptive modeling.

As has already been remarked, the interpretation of the genetic operators in economic terms is problematic, both from the point of view of realism and consistency. A number of decisions that are supposed to be internal to the firm are modeled exogenously in such a way as to abstract from the difficulties that real firms would have in carrying them out, in terms of both information acquisition and processing. In particular, there is no difficulty in inferring the genotype of firms, both because it is so simple and because there is no noise in the model. The absence of an adequately defined distinction between the genotype of the firm, the firm's model of the world, and the phenotype, the actions produced by that model, does nothing to clarify the genetic interpretation. Such a distinction is plainly more important when genotypes represent real objects and phenotypes real actions than when they are merely abstract representations of solutions to a problem. Furthermore, the representation of the firm's problem simply as the setting of an output decision continues the economic tradition of substituting the difficult issue of how agents develop models of their environment for the far simpler one of applying those models when they are correct.

Bearing in mind that the purpose of the fitness function in the GA is only instrumental—it is intended to make the process of reproduction exponentially efficient—it might be possible to make the model more realistic by describing the processes of bankruptcy (death) and imitation explicitly. This would dispense with the requirement for 'rational' imitation as successful firms should survive longer in the market to be imitated. The genotype of the firm would include a decision on whether to imitate based on data that were realistically available. Differential survival would thus be based on accumulated profit permitting firms to avoid bankruptcy despite noise and uncertainty in the environment. (Such a development might produce convergence results equivalent to the traditional GA, but that equivalence should not be taken for granted.)

Although the successful use of the GA to explain experimental data is a good example of independent verification of the analogy, the application of the model is insufficiently reflective about the implications of GA theory, or the choice of a suitable GA model. This is apparent from the fact that the results of the simulation could fairly confidently have been predicted from a knowledge of the GA and the problem space alone, independent of any economic interpretation. Most of the problem spaces considered are

nondeceptive and possess a single peak. Thus, except for additional realism in the mechanism for economic systems, which I have queried above, the GA has no real opportunity to display its robustness over and above traditional hill climbing algorithms. In fact, the use of the election operator to ensure convergence suggests both that the interdependence of profits is a problem for normal convergence, requiring a 'fix,' and that the problem space is simple enough to be amenable to hill climbing.

Finally, it appears that the motivation of model selection is at least partially theoretical, rather than empirical, in that the absence of convergence is regarded as a 'problem' to be 'resolved' by the introduction of the *ad hoc* election operator. (It is *ad hoc* because it changes the interpretation of the genotype. Previously the chromosomes represented actual decisions. In election there is an exogenous 'rational' comparison process being introduced into imitation which is not part of the genotype of individual firms. Furthermore, election operates over solutions generated by crossover and mutation that are themselves modeled exogenously. It is thus a 'second-order' operator.) It is also not clear whether this effective addition of hill climbing to the GA could prove counterproductive to convergence in more complex problem spaces. The need for convergence seems to be a value judgement imported from economics, as the experimental evidence for convergence is at best ambiguous. This view of the justification for the application of the model is consistent with the choice of an efficient formal GA over a more realistic description of the actual processes of imitation and persistence.

G7.1.5 Conclusions

The attempt to apply evolutionary computation to social processes can reveal important assumptions underlying both mathematical modeling of those processes and the instrumental use of evolutionary algorithms. Despite the critical tone of this analysis, Arifovic's work forms an important starting point for an evolutionary view of social action with considerable richness. In order for this potential to be realised, however, it is important that the differing requirements of descriptive and instrumental modeling are fully understood.

References

Alchian A A 1950 Uncertainty, evolution and economic theory *J. Polit. Econ.* **58** 211–22

Arifovic J 1990 *Learning by Genetic Algorithms in Economic Environments* Working Paper 90-001, Santa Fe Institute

——1994 Genetic algorithm learning and the cobweb model *J. Econom. Dynam. Control* **18** 3–28

Chattoe E 1994 The use of evolutionary algorithms in economics: metaphors or models for social interaction? *Multi-Agent Simulation and Artificial Life* ed E Hillebrand and J Stender (Amsterdam: IOS) pp 48–83

De Jong K 1992 Are genetic algorithms optimisers? *Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature (Brussels, 1992)* ed R Männer and B Manderick (Amsterdam: Elsevier) pp 3–13

Hodgson G M 1991 *Economics and Evolution: Bringing Life Back into Economics* (Cambridge: Polity)

Nelson R R and Winter S G 1982 *An Evolutionary Theory of Economic Change* (Cambridge, MA: Belknap–Harvard University Press)

Olivetti C 1994 *Do Genetic Algorithms Converge to Economic Equilibria?* Discussion Paper 24, University of Rome 'La Sapienza'

Rudolph G 1994 Convergence analysis of canonical genetic algorithms *IEEE Trans. Neural Networks* **NN-5** 96–101

G7.2 Intelligent hybrid systems for financial decision making

Suran Goonatilake

Abstract

This case study describes the use of genetic-fuzzy hybrid systems for supporting financial decision making. A novel architecture for inducing fuzzy rule-bases using genetic algorithms is presented. This combination of genetic algorithms and fuzzy logic produces easy to understand 'transparent' decision models that can be easily understood by technical personnel and high-level strategic decision makers alike. Although we discuss this approach with an example from the area of decision support in financial trading, this method evidently has wide applications in other areas of financial decision making including credit evaluation, corporate risk assessment, and insurance underwriting.

G7.2.1 Introduction

Intelligent systems are now being used to support decisions in tasks ranging from trading currency futures to predicting sales in supermarkets (Goonatilake and Treleaven 1995). While there is now an array of different types of intelligent techniques (neural networks, genetic algorithms, rule induction, etc.), each technique has particular strengths and limitations and cannot be successfully applied to every type of problem. For example, in a decision-making task that requires explicit explanations, neural networks are less applicable than a rule-induction approach. Similarly, for tasks that require constant adaptation and learning from the operating environment, a static expert system is far less useful than an adaptive method such as a neural network. Such limitations have been a central force in bringing about the combination of two or more intelligent techniques in such a way as to overcome the inherent limitations of individual techniques (Goonatilake and Khebbal 1995). It is these hybrid systems that are forming the basis of a new generation of intelligent decision-support systems.

In this case study we outline an intelligent hybrid-systems approach for financial decision making which combines *genetic algorithms* and *fuzzy logic*. The genetic algorithm is used to induce fuzzy decision rules operating on data with 'linguistic' categories such as *low*, *medium* and *high*. The genetic algorithm is based on Packard's genetic algorithm for complex data analysis (Packard 1990). The combination of genetic algorithms and fuzzy logic produces extremely easy to understand 'transparent' decision models which can be appreciated by technical personnel and high-level strategic decision makers alike. Furthermore, the induced decision models naturally lend themselves to judgmental revisions by decision makers.

B1.2, D2

G7.2.2 Packard's system for complex data analysis

Packard's genetic algorithm (Packard 1990) can be viewed as a *model-searching mechanism* that searches a very large space of possible models to find a good set of models that can capture underlying regularities of the given system being studied.

Assume the data to be a collection of pairs $(\boldsymbol{x}, y)$, where each $\boldsymbol{x}$ is a set of independent variables (*features)* and where y is the corresponding dependent variable (*classification variable*). Both the

independent and dependent variables must have discrete states, and if the source is continuous the values have to be discretized or 'binned'. The aim of the algorithm is to search for states of the independent variables, $\boldsymbol{x}$, which on average have a high correlation with particular desired states of y, the dependent variable. The induced patterns will take the form of a set of hypotheses or models, each being of the form *'when some subset of the independent variables satisfies particular conditions, a certain behavior of the dependent variable is to be expected*.' In a market forecasting context, the dependent variable will typically be a future (discretized) state of the system such as 'the market in 10 days (BUY or SELL)' and the independent variables will be (discretized) states of technical trading indicators such as 'Open interest *low*' and 'Volume *high*'.

The representation of models or *sets* in Packard's system is in the familiar *disjunctive normal form*, which specifies relationships between entities in terms of AND, OR relations. A conditional set or model contains as many 'condition positions' as there are independent coordinates, n, identifying each of them with one of the coordinates. Each 'condition position' will be allowed to take on either a value of *, indicating that no condition is set for the corresponding coordinate, or a sequence of numbers $(c_1, \ldots, c_k)$ indicating OR'ed values of the corresponding coordinate. For example,

$$(*, (5, 9), *, *, 7, *, *) \sim X_c$$

indicates that the conditional set X_c will be true if the second coordinate has a value of either 5 or 9, and the fifth coordinate has a value of 7. It will ignore the values of the other coordinates. If the aim of the algorithm is to find *good models* or conditional sets, then there must be a mechanism for evaluating the *goodness* or 'fitness' of a given model. This means finding the level of correlation between the states of the independent variables and the *target* dependent variable.

Let N_c be the total number of points in the conditional set X_c (the set of points that satisfy all the specified conditions). We then construct our empirical estimate of the conditional probability distribution of y values given the values of $\boldsymbol{x} \in X_c$:

$$P_c(y) = \frac{1}{N_c} \sum_{(x,y)\in X_c} \delta(y - y').$$

Packard (1990) has also introduced a 'devaluing' operator to guard against the building of conditional sets that have very small numbers of data points in them, and hence to reduce the effects of statistical flukes. A term proportional to $1/N_c$ is introduced here to achieve this devaluation.

The fitness F_c of a model or conditional set with the devaluation operation is therefore defined as

$$F_c(y) = P_c(y) - \frac{\alpha}{N_c}$$

where α is a parameter for adjusting the dependence on N_c.

We illustrate our implementation of Packard's system with the following example. Let there be three independent variables *max-speed*, *age-of-car* and *age-of-driver*; *max-speed* has three possible states, [low medium high], *age-of-car* has two possible states, [old new], and *age-of-driver* has three possible states, [young middle senior]. An empty list denoted as [] corresponds to the symbol * used by Packard. An example rule using these variables is:

```
[IF [max-speed [high]] AND [age-of-car [new]] AND [age-of-driver [young]] THEN [risk [high]]]
```

The genetic algorithm cycle has the following seven standard steps:

1. Initialization of a population of (random) rules.
2. Evaluation of fitness of each rule-base in the population.
3. Selection of parent rules for alteration.
4. Creation of new rules by crossover and mutation operators.
5. Deletion of the old rule population.
6. Creation of a new population by inserting altered rules and the fittest rules.
7. Go to 3 until a satisfactory rule(s) is found or a specified number of iterations have been completed.

The crossover operation swaps the conditions of rules with other conditions of other rules, at the same conditional locations. The crossover rate (C_r) which is set by the user determines the probability of a crossover operation occurring at a particular conditional point.

If there are two population members (chromosomes) c1, c2,

```
c1:  IF [max-speed [high]] AND [age-of-car [new]] THEN [risk [high]]
c2:  IF [max-speed [low]] AND [age-of-car [old]] THEN [risk [low]]
```

then the effect of crossover can result in formation of the following two new rules c3 and c4:

```
c3:  IF [max-speed [low]] AND [age-of-car [new]] THEN [risk [low]]
c4:  IF [max-speed [high]] AND [age-of-car [old]] THEN [risk [high]]
```

There are three mutation operators in the system. The probability of a mutation operation being applied is determined by the user-specified mutation rate M. The three mutation operators are:

1. Picking a new coordinate
   `age-of-driver []` → `age-of-driver [young]`
2. Deleting a coordinate
   `age-of-driver [young]` → `age-of-driver []`
3. Changing the value of a coordinate
   `age-of-driver [young]` → `age-of-driver [senior]`

G7.2.3 Fuzzy data preprocessing

Most decision makers in finance and business commonly use linguistic categories (e.g. low, high, large) to describe complex relationships in their domain. We therefore ideally need a mechanism to convert 'raw' data from a domain (e.g. price, volume and open interest data) into such linguistic symbolic descriptions. We use a relatively simple method based on the use of a clustering algorithm to convert such data into linguistic descriptions. It is on these linguistic descriptions that the genetic algorithm operates.

The starting point for the preprocessing method is for the user to specify linguistic 'labels'. These labels are for the symbolic categories into which the algorithm will subsequently classify raw data. Examples of these labels are low, medium, high, and small, moderate and big. The linguistic categories should be specified in an increasing order, such as low, medium, high.

Once the order of the labels is specified, a clustering algorithm is applied to the raw market data. The clustering algorithm used is the single-linkage clustering method (SLINK). A public domain implementation of the SLINK algorithm (Stolcke 1992) is used for all clustering operations.

A heuristic cluster selection algorithm is applied to the cluster tree to select clusters whose ranges roughly correspond to linguistic labels specified by the user. The cluster selection algorithm operates on the heuristics that the distribution of the data points among the different categories is roughly equal and that their data ranges are also similar. Details of this cluster selection algorithm can be found in Goonatilake and Feldman (1994). The numerical ranges of the clusters corresponding to the linguistic categories are then used to classify unseen data items.

For example, in the case depicted in figure G7.2.1, the algorithm has chosen the *medium* cluster with a range between 3100 and 4300, and the *high* cluster with values between 4700 and 5000. An unseen data item which has a value of 3300 will be classified as being medium and a value of 4900 will be classified as being high.

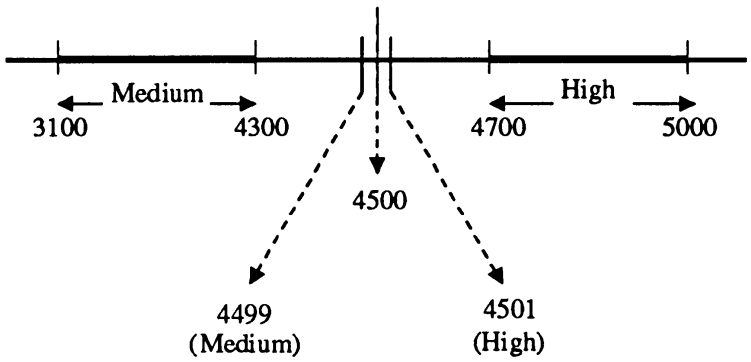


Figure G7.2.1. The class boundaries.

G7.2.3.1 Defining the fuzzy sets

The cluster selection algorithm produces clusters whose boundaries are 'crisp' where one linguistic category has a sharp jump to another linguistic category. We now 'smooth' these boundaries to produce fuzzy descriptions. We achieve this by defining triangular *fuzzy membership functions* using the midpoints of the selected cluster ranges as 'anchor points' (see figure G7.2.2). D2.1

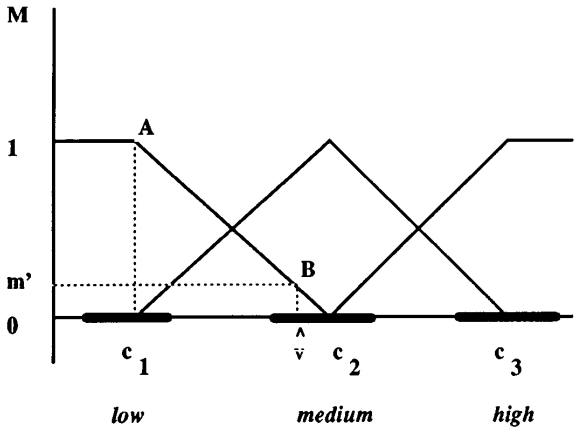


Figure G7.2.2. Fuzzy set membership.

G7.2.3.2 Defining membership functions for the rule consequents (decisions)

The membership functions for the consequents, the decisions, (e.g. buy or sell) are defined heuristically. In the financial trading example (detailed in section G7.2.5), the trading decisions are defined as having a range of $[-3, +3]$ where the negative values indicate a SELL decision while the positive values indicate a BUY decision (see figure G7.2.3). A membership function, DO-NOTHING, reflecting the decision not to trade has also been defined. The numerical values indicate the level of confidence of the decision (e.g. -2.9 indicates a very definite SELL decision, while -0.8 indicates a less definite SELL decision).

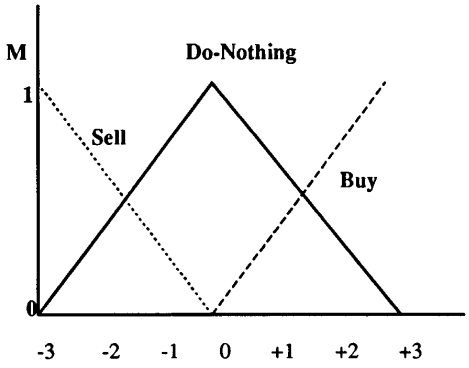


Figure G7.2.3. Membership function for trading decisions.

G7.2.4 Inducing fuzzy rule-bases

An inherent limitation of fuzzy systems is that the rules have to be manually specified by a domain expert. This is often a time-consuming and expensive process that should ideally be automated. Here we describe the use of a genetic algorithm to find rules for fuzzy systems. The system is *function-replacing hybrid* according to the hybrid-systems classification scheme presented by Goonatilake and Khebbal (1995). Function-replacing hybrids are hybrids where a principal function of a given intelligent technique (e.g. weight updating, rule specification) is *replaced* by another intelligent technique.

The genetic algorithm cycle is as follows. The population is first initialized with a random collection of fuzzy rule-bases. At each iteration, each fuzzy rule-base makes fuzzy inferences on fuzzified data. The final results are then passed through a threshold and the final trading decisions are obtained. These decisions are then compared with the known 'best' decisions using the past data, and the fitness of rule-bases are calculated accordingly. The fuzzy rule-bases are ranked in terms of their fitness, and afterwards mutation and crossover operations are performed to produce new rule-bases. Over time this procedure produces a collection of highly effective fuzzy rule-bases.

An example of two members, GF1 and GF2, of a rule population (each rule-base having four rules) is:

```
[GF1   [ma-diff-1-20-fuzzy [positive]] AND [oi-rsi-14-fuzzy [high]] THEN [action [BUY]]
[ma-diff-1-20-fuzzy [negative]] AND [oi-rsi-14-fuzzy [low]] THEN [action [SELL]]
[ma-diff-1-20-fuzzy [positive]] AND [oi-rsi-14-fuzzy [medium]] THEN [action [BUY]]
[ma-diff-1-20-fuzzy [positive]] AND [oi-rsi-14-fuzzy [high]] THEN [action [BUY]]]

[GF2   [ma-diff-1-20-fuzzy [positive]] AND [oi-rsi-14-fuzzy [high]] THEN [action [SELL]]
[ma-diff-1-20-fuzzy [negative]] AND [oi-rsi-14-fuzzy [high]] THEN [action [SELL]]
[ma-diff-1-20-fuzzy [neutral]] AND [oi-rsi-14-fuzzy [low]] THEN [action [BUY]]
[ma-diff-1-20-fuzzy [neutral]] AND [oi-rsi-14-fuzzy [high]] THEN [action [BUY]]]
```

The evaluation of each member is performed as follows. We apply a compositional rule of inference (Mamdani and Assilian 1975) and use the *centre of area* method (Barenji 1992) as the defuzzification procedure to obtain the final result. As defined by the universe of discourse of trading decisions, this result will have a range $[-3, +3]$ (values close to -3 indicate a SELL decision, values closer to $+3$ indicate a BUY decision). We then apply a threshold to infer the final decision (values < -2.2 SELL, values $> +2.2$ BUY).

After this, Packard's fitness evaluation procedure (as detailed in section G7.2.2) is applied and the fitnesses of the rule-bases are computed. That is, for each fuzzy rule-base the fitness of its decisions are calculated where N_c are data entries that return values beyond the specified fuzzy threshold (values < -2.2 SELL, values $> +2.2$ BUY).

G7.2.5 The financial trading application

We now investigate the application of the above approach as a mechanism for supporting decisions in the domain of currency trading. The method is used to create fuzzy rule-bases that operate on technical trading indicators (Kaufman 1987). Technical trading rules are used by a large number of traders and our method provides an automated method for discovering such trading knowledge. We do not envisage that this type of approach will completely *automate* the trading process, but instead take the view that it is a good *decision-support* tool for traders. Traders may overrule or change the conclusions of the rules due to considerations external to the models (e.g. political events). Therefore, ideally, rules discovered by this approach will be firstly presented to a trader for judgmental revisions before being used for trading.

The moving-average method is a commonly used technical trading indicator. Generally two moving averages are used—a long period (e.g. the moving average of the last 200 days' prices) and a short period (e.g. the moving average of the last 10 days' prices); this is typically written as a 10–200 system. A 1–10 system would be one in which the long moving average was for a 10 day period and the short was the actual daily price. The general idea behind computing the moving averages is that they smooth the generally volatile time series, and provide an indication of the general trend of the market (Kaufman 1987).

The moving average of prices is given by

$$\mathrm{MA}_t = \frac{1}{N}\sum_{i=0}^{N-1} P_{t-i}$$

where N is the number of days, P is the price and MA_t is the moving average on day t.

There are several ways of using moving averages for making trading decisions. One type of trading strategy is to execute BUY trades when the short moving average is higher than the long moving average, and to execute SELL trades when the short moving average is lower than the long moving average.

The rules corresponding to these hypotheses are:

(E1) If the *short moving average* is higher than the *long moving average* then the market is likely to *rise* (action: BUY).

(E2) If the *short moving average* is lower than the *long moving average* then the market is likely to *fall* (action: SELL).

A variation of this approach is to execute trades when the moving averages cross each other. With this strategy a BUY trade is executed at the point when the short moving average becomes higher than the long moving average, and a SELL trade is executed at the point when the short moving average becomes lower than the long moving average.

The rules corresponding to these hypotheses are:

(E3) If the *short moving average* crosses the *long moving average* from *below* then the market is likely to *rise* (action: BUY).

(E4) If the *short moving average* crosses the *long moving average* from *above* then the market is likely to *fall* (action: SELL).

The effect of these rules can be seen in figure G7.2.4.

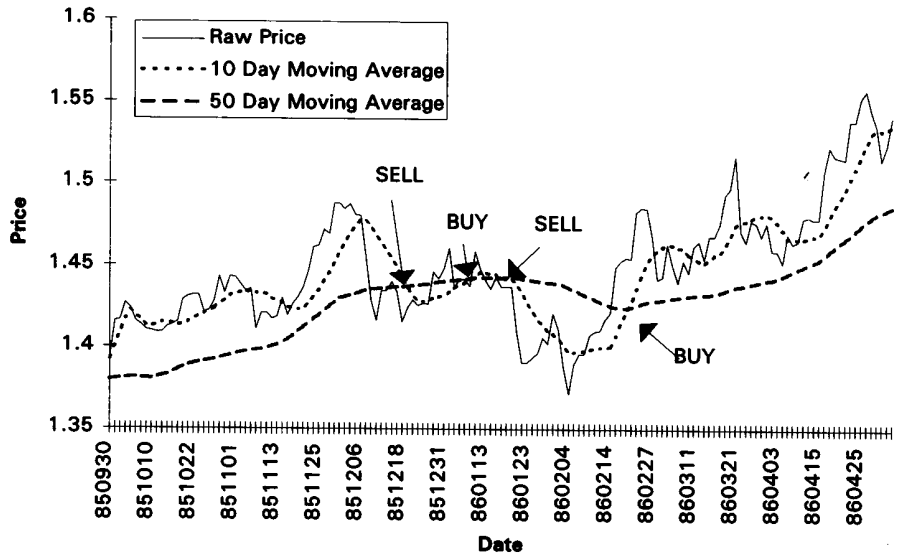


Figure G7.2.4. Trading signals from moving-average model.

All the above moving-average schemes are essentially based on measures reflecting the *difference* between the values of the two moving averages. We have used a simple measure of this difference,

$$\mathrm{MA}_{\mathrm{diff}} = \frac{\mathrm{SMA} - \mathrm{LMA}}{\mathrm{SMA}}$$

where SMA is the short moving average, LMA is the long moving average and $\mathrm{MA}_{\mathrm{diff}}$ is the measure of difference between the two moving averages. A possible trading strategy based on this difference measure is:

- If the $\mathrm{MA}_{\mathrm{diff}}$ is *positive* then the price is likely to *rise* (action: BUY).
- If the $\mathrm{MA}_{\mathrm{diff}}$ is *negative* then the price is likely to *fall* (action: SELL).

The aim of this work is to use the genetic algorithm to discover fuzzy trading rules similar to the ones described above.

The data used are prices of the British pound against the US dollar. The independent variables are moving-average differences of the closing prices, volume and open interest, and the dependent variable is the price after 10 days (BUY or SELL). Data from 1982 to 1984 are used to derive the cluster ranges while data from 1984 to 1987 are used to induce the rules. Data from 1987 to 1988 are used as a validation set while data from 1988 to 1992 are completely unseen data used for testing.

We follow Weiss and Kulikowski (1991) and attempt to avoid overfitting the training data by monitoring classification rates on the training and validation sets. At the classification error 'turning points' the fittest fuzzy rule-base is selected, and then applied to completely unseen data.

The following are the fittest two fuzzy rule-bases (FR1, FR2) induced using the British pound data. Each rule-base consists of four fuzzy rules.

```
[[FR1   [G1[ma-diff-1-50-fuzzy-values []] AND [ma-diff-1-100-fuzzy-values []]
AND [ma-diff-1-200-fuzzy-values [negative]] AND [vol-ma-diff-1-10-fuzzy-values []]
AND [vol-ma-diff-1-20-fuzzy-values []] AND [oi-ma-diff-1-10-fuzzy-values []]
AND [oi-ma-diff-1-20-fuzzy-values [neutral]] AND [price-fuzzy-vola-20 []]
AND [price-fuzzy-vola-50 [high]] AND [price-fuzzy-vola-100 []]
THEN [action [SELL]]]

[G2[ma-diff-1-50-fuzzy-values []] AND [ma-diff-1-100-fuzzy-values []]
AND [ma-diff-1-200-fuzzy-values [negative]] AND [vol-ma-diff-1-10-fuzzy-values []]
AND [vol-ma-diff-1-20-fuzzy-values []] AND [oi-ma-diff-1-10-fuzzy-values []]
AND [oi-ma-diff-1-20-fuzzy-values [neutral]] AND [price-fuzzy-vola-20 []]
AND [price-fuzzy-vola-50 []] AND [price-fuzzy-vola-100 [medium]]
THEN [action [SELL]]]

[G3[ma-diff-1-50-fuzzy-values []] AND [ma-diff-1-100-fuzzy-values []]
AND [ma-diff-1-200-fuzzy-values [negative]] AND [vol-ma-diff-1-10-fuzzy-values []]
AND [vol-ma-diff-1-20-fuzzy-values []] AND [oi-ma-diff-1-10-fuzzy-values []]
AND [oi-ma-diff-1-20-fuzzy-values [negative]] AND [price-fuzzy-vola-20 []]
AND [price-fuzzy-vola-50 []] AND [price-fuzzy-vola-100 []]
THEN [action [SELL]]]

[G4[ma-diff-1-50-fuzzy-values []] AND [ma-diff-1-100-fuzzy-values []]
AND [ma-diff-1-200-fuzzy-values [negative]] AND [vol-ma-diff-1-10-fuzzy-values []]
AND [vol-ma-diff-1-20-fuzzy-values []] AND [oi-ma-diff-1-10-fuzzy-values []]
AND [oi-ma-diff-1-20-fuzzy-values [negative]] AND [price-fuzzy-vola-20 [low]]
AND [price-fuzzy-vola-50 []] AND [price-fuzzy-vola-100 []]
THEN [action [SELL]]]

[FR2   [G1[ma-diff-1-50-fuzzy-values []] AND [ma-diff-1-100-fuzzy-values []]
AND [ma-diff-1-200-fuzzy-values [negative]] AND [vol-ma-diff-1-10-fuzzy-values []]
AND [vol-ma-diff-1-20-fuzzy-values []] AND [oi-ma-diff-1-10-fuzzy-values []]
AND [oi-ma-diff-1-20-fuzzy-values [neutral]] AND [price-fuzzy-vola-20 []]
AND [price-fuzzy-vola-50 []] AND [price-fuzzy-vola-100 [high]]
THEN [action [SELL]]]

[G2[ma-diff-1-50-fuzzy-values []] AND [ma-diff-1-100-fuzzy-values []]
AND [ma-diff-1-200-fuzzy-values [negative]] AND [vol-ma-diff-1-10-fuzzy-values []]
AND [vol-ma-diff-1-20-fuzzy-values []] AND [oi-ma-diff-1-10-fuzzy-values []]
AND [oi-ma-diff-1-20-fuzzy-values [neutral]] AND [price-fuzzy-vola-20 []]
AND [price-fuzzy-vola-50 [low]] AND [price-fuzzy-vola-100 []]
THEN [action [SELL]]]

[G3[ma-diff-1-50-fuzzy-values []] AND [ma-diff-1-100-fuzzy-values []]
AND [ma-diff-1-200-fuzzy-values [negative]] AND [vol-ma-diff-1-10-fuzzy-values []]
AND [vol-ma-diff-1-20-fuzzy-values []] AND [oi-ma-diff-1-10-fuzzy-values []]
AND [oi-ma-diff-1-20-fuzzy-values [negative]] AND [price-fuzzy-vola-20 [high]]
AND [price-fuzzy-vola-50 []] AND [price-fuzzy-vola-100 []]
THEN [action [SELL]]]

[G4[ma-diff-1-50-fuzzy-values []] AND [ma-diff-1-100-fuzzy-values []]
AND [ma-diff-1-200-fuzzy-values [negative]] AND [vol-ma-diff-1-10-fuzzy-values []]
AND [vol-ma-diff-1-20-fuzzy-values []] AND [oi-ma-diff-1-10-fuzzy-values []]
AND [oi-ma-diff-1-20-fuzzy-values [negative]] AND [price-fuzzy-vola-20 []]
AND [price-fuzzy-vola-50 []] AND [price-fuzzy-vola-100 [high]]
THEN [action [SELL]]
```

The system produced 61% correct trades. We also undertook a detailed study of assessing human trader performance in the same foreign exchange trading task (details of this assessment are beyond the scope of this paper). The human trader was correct for 64.2% of the time.

Ideally the fuzzy rule-bases generated by the genetic algorithm should be revised judgmentally by a domain expert. As the model does not contain information external to it (e.g. political events), a trader may examine the rules and change any conditions if he or she so wishes.

Although we have demonstrated this approach as a mechanism for discovering knowledge in financial trading, it evidently has many applications in several other domains where explicit and easy to understand explanations of decision models is a prime concern. We are currently evaluating this approach in the areas of credit evaluation, insurance risk assessment, and credit card fraud detection.

Acknowledgement

This article was first published in the Proceedings of the 1995 ACM Symposium on Applied Computing, held in Nashville, TN, on 26–28 February 1995. Copyright 1995 Association for Computing Machinery. Republished by permission.

References

Barenji H 1992 Fuzzy logic controllers *An Introduction to Fuzzy Logic Applications in Intelligent Systems* ed R Yager and L Zadeh (Dordrecht: Kluwer Academic)

Goonatilake S and Feldman K 1994 Genetic rule induction for financial decision making *Genetic Algorithms in Optimisation, Simulation and Modelling* ed J Stender, E Hillebrand and J Kingdon (Amsterdam: IOS Press)

Goonatilake S and Khebbal S (ed) 1995 *Intelligent Hybrid Systems* (New York: Wiley)

Goonatilake S and Treleaven P (ed) 1995 *Intelligent Systems for Finance and Business* (New York: Wiley)

Kaufman P J 1987 *The New Commodity Trading Systems and Methods* (New York: Wiley)

Mamdani E and Assilian S 1975 An experiment in linguistic synthesis with a fuzzy logic controller *Int. J. Man-machine Studies* **7** 220–31

Packard N H 1990 A genetic learning algorithm for the analysis of complex data *Complex Systems* **4** 543–72

Stolcke A 1992 *Cluster Program Manual* University of Colorado

Weiss S M and Kulikowski C A 1991 *Computer Systems that Learn: Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning and Expert Systems* (San Mateo, CA: Morgan Kaufmann)

G8.1 Learning and upgrading rules for an optical character recognition system using genetic programming

David Andre

Abstract

Rule-based systems used for optical character recognition (OCR) are notoriously difficult to write, maintain, and upgrade. This case study describes a method for using genetic programming (GP) to automatically generate and upgrade rules for an OCR system. Sets of rules for recognizing a single character are encoded as LISP programs and are evolved using GP. The rule sets are programs that evolve to examine a set of preprocessed features using complex constructs including iteration, pointers, and memory. The system was successful at learning rules for large character sets consisting of multiple fonts and sizes, with good generalization to test sets. In addition, the method was found to be successful at updating human-coded rules written in C for new fonts. This research demonstrates the successful application of GP to a difficult, noisy, real-world problem, and introduces GP as a method for learning sets of rules.

G8.1.1 Project overview

The goal of optical character recognition (OCR) is to automatically translate scanned pictoral images of printed documents into text documents. Text documents consume much fewer resources than do pictoral images and allow for easy handling of the documents for word processing and for automatic classification, retrieval, and storage systems. Many OCR systems are based on many sets of rules, where each set of rules describes a single character across all fonts. The rule set must capture the characteristics of the character that distinguish it from all other characters. Creating and testing these rule sets by hand is notoriously difficult, because any changes in a rule set must be tested on a very large number of character sets to ensure that the rule set accepts all examples of the desired character and rejects all others.

This case study presents a system that can either learn rule sets for characters from scratch or can upgrade rule sets that were initially created by hand. Starting from an initial population of rule sets encoded as LISP programs that were either randomly generated or derived from human-coded sets, we automatically generated better rule sets through the process of *genetic programming*. Genetic programming (GP) (Koza 1992) is an extension to the basic *genetic algorithm* where the entities undergoing evolution are computer programs represented as LISP-like parse trees. Rule sets in each subsequent generation are created by simulated analogs of natural crossover and reproduction. Figure G8.1.1 presents a graphical overview of the process of GP.

B1.5.1 B1.2

G8.1.2 Design process

G8.1.2.1 The recognition system without genetic programming

One of the difficulties in OCR is the sheer number of bits of information present in every character (1225 bits on average for printed text at 300 dpi). Often, prior to the evaluation of rule sets in many systems,

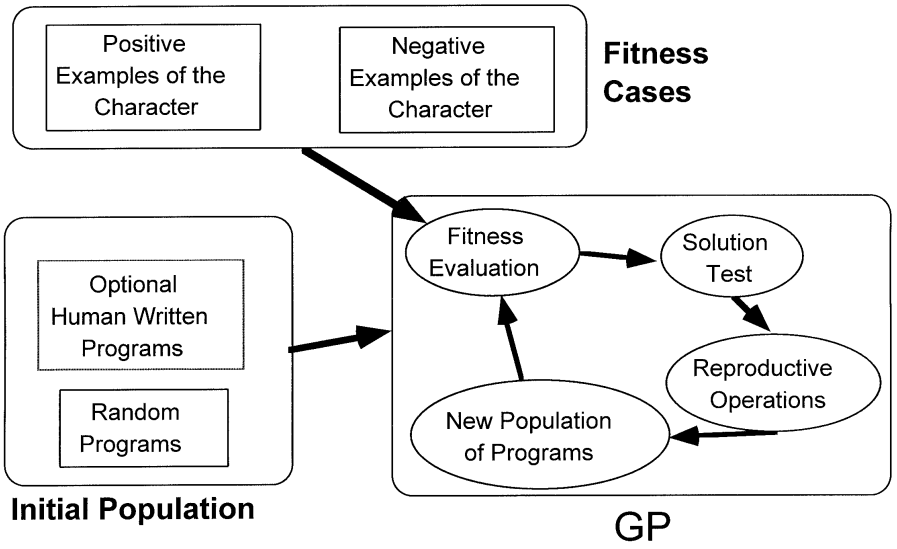


Figure G8.1.1. Graphical overview of the GP system.

the character images are simplified into combinations of more general features. This simplification allows simple rules to specify a character, but also loses some information about the character, as the extraction of very general features such as lines or curves is a many-to-one transformation. One attempted solution has been to use features of the bitmap that allow reconstruction of the character image. Following this approach, the system presented in this research uses the outline of the character as its features (i.e. a feature breakdown that contains pixel information for only the boundaries of the character).

The first step in the recognition of any character is to extract the boundary pixels from the character bitmap. This is done using a quick one-pass raster scan method (Andre 1993) that provides output in the form of a bidirectional circular linked list for the boundary of the character and for each interior hole. Each element in a list contains row and column information for a boundary pixel. The outer boundary of the character is then split into four segments (top, left, right, and bottom) using a technique that is robust to moderate levels of noise (Amos 1993). An example of this segment breakdown is shown in figure G8.1.2. At this point, simple bounding box values (i.e. the maximum and minimum row and column) are calculated for each hole, for each segment, and for the character as a whole. In addition, the number of pixels in each hole boundary is stored, and the segments are ranked according to their number of pixels. Thus preprocessing produces a simplified packet of information consisting of a number of linked lists and a number of simple statistics for each list.

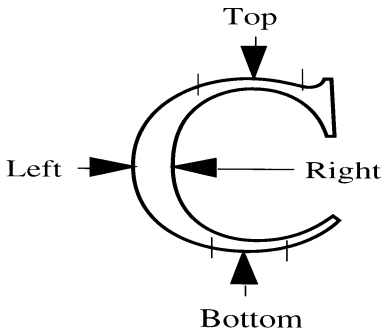


Figure G8.1.2. Example segmentation of a character boundary. A pre-processor robustly calculated these segments for each character using a hand-crafted contour-based method.

The recognition system contains a rule set for each character that can be found in English printed material. Each rule set contains rules that express relations among values in the simplified packet of information produced by preprocessing. These rules are then combined together through logical primitives

such that the entire rule set will return true only for the desired test character. If two rule sets do fire for the same test character, then a response is chosen by likelihood of occurrence of the characters, but an uncertainty flag is raised indicating that the confidence of this classification is low. Examples of rules are given in table G8.1.1. Examples in C and in the GP LISP language are given in table G8.1.3, in section G8.1.3. These rules may include complicated loops and multiple operations on the linked lists of pixels (rules 4, 5, table G8.1.1) or they may simply be quantitative statements of simple bounding box statistics (rules 1–3, table G8.1.1).

Table G8.1.1. Examples of rules for the letter 'C'.

1.	Right segment (Rseg) is longer than any other.
2.	Number of rows in Tseg is less than 10% of those in Lseg.
3.	The middle column of Lseg is to the left of either edge of Lseg.
4.	Starting at the intersection of Tseg and Rseg, the top point on Rseg near the middle column is reached before reaching the middle row and after a downward spike.
5.	On the lower half of Rseg, between the lowest point in the inner curve and the maximum column point, there is no point at which there is a run of four vertical pixels, nor any place where the boundary doubles back in the horizontal direction.

G8.1.2.2 Motivations for using genetic programming

Creating the rule sets for this recognition system by hand is no trivial task. Given that the goal was robust recognition of several fonts, creating a rule set for a character required many iterative steps of writing, testing, and modifying the rule sets. Thus, some automated method of creating these sets of rules is desired. *Classifier systems* (Holland and Reitman 1978, Smith 1980) provide one method of learning rules or rule sets; however, they have limitations that were problematic for our approach. Although our rules could be easily expressed as if–then rules, both the if and then components of the rules consist of extremely complex computational steps. In addition, rules in a rule set were often heavily interconnected, a feature which we found lacking in both the Michigan approach (Holland and Reitman 1978) to classifier systems (a population of fixed-length rules) and the Pittsburgh approach (Smith 1980) (a population of fixed-structure rule sets). It was certainly possible to represent this problem in terms of traditional classifier systems, but because of the desire to translate between the learning system and our hand-coded C programs, a system where the entities that were learned were more similar in nature to C programs was preferred. GP (Koza 1992) provides such a system. In this alternative approach to learning sets of rules, the rules are evolved as part of a computer program. The rules themselves, as well as the interrelations between rules, can be complex. Both the rules and the method for combining them are evolved in GP.

B1.5.2

GP also seemed promising because it had been successful on a variety of classification problems. Koza (1993) evolved programs that could recognize an 'L' and an 'I' using coevolved Boolean templates and control code for moving the templates. Andre (1994a) extended this work by evolving programs that were successful at recognizing low-resolution digits using coevolved two-dimensional feature detectors. GP has also been successfully applied to problems in *machine vision* (Teller and Veloso 1995, Tackett 1994) and *biological classification problems* (Handley 1994, Koza 1994, Koza and Andre 1996a, b).

G8.2 G6.1

G8.1.3 Preparatory steps and implementation

There were two motivations for choosing the programmatic ingredients that make up the programs in the population. First, the primitives had to provide the necessary power to solve the problem. Second, they had to be capable of expressing the hand-coded versions of the rules. Complicated functions to simulate looping, pointers, storage, and data access were thus included in the function set. These ingredients, shown in table G8.1.2, were capable of expressing all 150 of the hand-coded rule sets. Table G8.1.3 shows C and GP tree versions of two rules, which were among those translated into GP tree form for the letter 'C'.

The fitness of a given program in the population is determined by its ability to classify a number of character bitmaps, which are the fitness cases (or the training cases) for the problem. The exact number of bitmaps varies for each experiment. In each experiment, the set of fitness cases is split into positive and

Table G8.1.2. Programmatic ingredients that make up the evolving programs in GP.

Terminals:		
General:	first_row, first_col, last_row, last_col	These correspond to the bounding box statistics for the entire character.
For each Hole(2):	NumPixels, first_row, first_col, last_row, last_col.	These correspond to the bounding box statistics and the number of pixels for the holes.
For Each Segment(4):	Rank, first_col, last_col, first_row, last_row.	These correspond to the bounding box statistics and the rank for the linked-lists of the segments.
Constants:	random_int, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.	Simple constants. The random integer is an ephemeral random constant.
Functions:		
Arithmetic	add(**a**,**b**), minus(**a**,**b**), times(**a**,**b**), divide(**a**,**b**)	The simple mathematical functions. Protected division returns **a** 1 if the **b** is 0.
IF – Program Structure	LessThanElse(**a**,**b**,**c**,**d**)	If **a** <= **b** then executes **c**, else executes **d**.
IF – Program Structure	LessThanReturn(**a**,**b**,**c**)	If **a** <= **b** then execute **c**, else return 0.
Feature Access	VHits(**a**,**b**)	This returns the number of times the segment specified by (**a** mod 4) crosses the row **b**.
Feature Access	HHits(**a**,**b**)	This returns the number of times the segment specified by (**a** mod 4) crosses the row **b**.
Pointer Update	goto(**a**)	Moves the Current-Pointer to the start of the segment specified by (**a** mod 4).
Pointer and Program Structure	GotoandDo(**a**,**b**)	Moves the Current-Pointer to the start of the segment specified by (**a** mod 4), executes **b**, and then returns the pointer to its previous location.
Iterative Structure	GoForwardUntil(**a**,**b**,**c**,**d**)	Moves the Current-Pointer forward until **a** <= b, or until the end of the segment is reached. Then, if **a** <= **b**, executes **c**, else executes **d**.
Iterative Structure	GoBackwardUntil(**a**,**b**,**c**,**d**)	Like GoForwardUntil, except backwards.
Program Structure	Progn(**a**,**b**)	-Executes **a**, then executes and returns **b**.
Memory Access	SetX(**a**)	-Sets a storage variable to **a**.
Memory Access	GetX()	-Returns the value of the storage variable.
Pixel Access	CurrentRow(), CurrentCol()	-Returns the current value for row or column.
Pixel Access	(Row/Column) (Forward/Backward) FromStart(**a**)	-Returns the row/column of the pixel that is **a** steps forward/backward from the start of the current segment.
Pixel Access	(Row/Column) (Forward/Backward)FromEnd(**a**)	Same as above for the end of the segment.
Pixel Access	(Row/Column) (Forward/Backward) FromCurrent(**a**)	Same as the above for the current pointer.

negative cases. The positive set of fitness cases consisted of bitmaps of the letter ‘C’ in several different fonts and sizes and the negative fitness cases consisted of bitmaps of various other characters. To score perfectly, an individual must return a value greater than zero when tested on a ‘C’, and must return a value equal to or less than zero when tested on any other character. Penalties are assessed for incorrect responses. If an individual mistakenly classifies a bitmap as a ‘C’, the individual is penalized for a false positive. If an individual mistakenly fails to classify a bitmap as a ‘C’, the individual is penalized for a false negative. For further information about implementation details, see the article by Andre (1994b). The penalties for each experiment are shown in table G8.1.4.

Table G8.1.3. Two examples of C and GP tree versions of rules, which were among those translated into GP tree form for the letter 'C'. The first is quite simple; it is true if and only if the right segment is the longest segment. The second rule is more complicated. The rule first finds the pixel on the left segment that is halfway down. Then, it returns true if this pixel is to the left of both the start and end of the left segment. In a sense, this rule insures that the curvature of the left segment matches that of a 'C'. The left segment of an 'A', for example, fails this test because the midway pixel is to the right of the lower end of the left segment. These rules are clearly human designed, but they are similar in complexity to those evolved by GP.

In C: (cptr is a pointer to a pixel)

```
(1)  if (Rseg→rank != 1) return(0);
(2)  cptr = Lseg→Start;
     mrow = (Lseg→first_row + Lseg→last_row)/2;
     while (cptr != Lseg→End && cptr→row < mrow)
          cptr = cptr→forward;
     if (Lseg→End→col < cptr→col) return(0);
     if (Lseg→Start→col < cptr→col) return(0);
```

In GP-tree form:

```
(1)  (LessThanReturn 1 Rseg_rank (LessThanReturn Rseg_Rank 1 1))
(2)  (GotoAndDo 2
          (Progn (GoForwardUntil (divide
                                     (plus Lseg_first_col Lseg_last_col) 2)
                                     (CurrentRow) 1 1)
                 (LessThanReturn
                          (CurrentCol)
                          (GotoAndDo 2 (ColForwardFromStart 0))
                          (LessThanReturn (CurrentCol)
                                     (GotoAndDo 1 (ColBackwardFromEnd 0))
                                     1))))
```

The final step in preparing to use GP is to determine the values for the various run parameters, and to determine the termination criteria for each run. Initial random programs were allowed to be only six levels deep, and were either random (experiment 1—learning from scratch), or a mix of random and seeded individuals (experiment 2—updating existing rule sets). Evolved programs were limited to a depth of 20. *Tournament selection* with a tournament size of eight was used. The percentage of the genetic operations was as follows: unmodified reproduction (i.e. copy to the next generation) 10%, crossover 75%, mutation 15%. The mutation operation replaced a randomly chosen subtree in an individual with a randomly generated subtree. Runs were terminated if an individual with perfect (zero) fitness was found, or after the maximum number of generations. C2.3

All experiments were run using a modified version of the *Simple Genetic Programming Code*, written by Walter Tackett (1993). The code was modified to run on a network of seven 486 PCs running Microsoft NT. The creation and breeding of the single panmictic population was performed on a single workstation; individuals were then sent to each of the other workstations only for the fitness calculation and the individuals' fitnesses were then reported back to the breeding workstation. The character bitmaps used in this study were obtained by scanning printed characters in binary format.

Table G8.1.4. Fitness variables and fitness case counts for the two experiments.

Experiment	No of positive cases	No of negative cases	False positive penalty	False negative penalty
1	20	600	1	30
2	3002	7000	3	7

G8.1.4 Details and results of experiments

G8.1.4.1 Experiment 1: learning rule sets from scratch

The purpose of experiment 1 was to attempt to learn solutions for the 'C' that would generalize well to never-before-seen test data. Thus, a moderately large training set was used, consisting of one or more examples of each character from Helvetica, Courier, and Times Roman of sizes 8, 10, and 12. There were 20 positive examples and 600 negative examples. The population size was 8000; run times varied from three to eight days. The maximum number of generations was 200 and the initial populations were random.

All of the eight runs that were performed produced solutions that scored above 99% on the training set, and five of the runs produced solutions scoring 100% on the training set. One successful run produced a 100% correct solution on generation 180 of a run lasting six days. Over many pages of test data, it was found to correctly classify between 96 and 99% of the characters on a page. Human-coded rule sets for characters typically score 99% on such test sets; the performance of GP on this problem is thus only slightly less than human performance. As is common in GP, the exact nature of the solution was difficult to ascertain, but it appeared in many respects to resemble the hand-coded rule set. For example, the predominant structure in both the evolved and the hand-coded individual was a set of nested 'LessThanReturn' functions.

G8.1.4.2 Experiment 2: upgrading an existing rule set

This experiment was designed to test the system's ability to upgrade an existing rule set to handle a new font. A human-written rule set for the letter 'C' was translated by hand into the GP tree language, and was tested to have identical performance to the C language version. The rule set for the 'C' was tested on many new fonts, where it was found to perform badly on the Eurostyle font. The 'C's in that font were often rejected, and many of the 'G's in that font were accepted as 'C's. One possible explanation for this is that the 'G's in Eurostyle are very similar to the 'C's; the 'G' lacks the horizontal crossbar that is so characteristic of the 'G' in other fonts. A set of fitness cases was developed that captured the fonts that the hand-coded version of the 'C' could accurately classify. This set included 3000 positive and 7000 negative cases. This set contained many different fonts with examples from at least five different sizes in each font. To attempt to upgrade for the Eurostyle font, two examples of a Eurostyle 'C' were added to the positive fitness cases. Thus, there were 3002 positive cases, and 7000 negative. The initial population consisted of the translated individual, each of the separate rules found in the translated individual, and random individuals. The population size was 5000 and run times varied from several hours to several days. The maximum number of generations was 100.

A representative successful individual was found on one of the three runs in generation 12 (the other runs were also successful, but took longer to produce a successful individual). The individual was tested on several pages of multiple sizes of Eurostyle font, and was found to be perfect at distinguishing a 'C' from a non-'C'. Even though the individual had only been trained on two examples of one size, the solution generalized to many sizes. Further analysis indicated that the upgraded individual was very similar to the hand-coded individual, but with a change in the part of the code that discriminated between a 'C' and a 'G'—the rule set had evolved to accept Eurostyle 'C's and reject Eurostyle 'G's.

G8.1.5 Conclusions

The results from the experiments described in this paper indicate that a GP OCR system can successfully evolve general rule sets that accurately classify printed characters. The letter 'C' was chosen for its high number of 'lookalikes', and thus it can be expected that rule sets would be at least as easy to learn for the other characters. While it would take a long time to learn a generalizable rule set for all characters, the experiments presented here suggest that GP is capable of it. Parallel processing approaches in genetic programming (Andre and Koza 1996) also provide sufficient computational power to allow rule sets to be learned in hours, rather than in weeks. Conceivably, the GP OCR system could be used to generate 96–99% accurate rule sets in a fraction of the time it would take human programmers. In addition, it was shown that a GP OCR system can successfully update hand-coded rule sets to handle new fonts. After updating, these rule sets can be translated back into C so they could easily be integrated with human-coded rule sets.

There are several limitations inherent in the current work that require additional research. Available computer time restricts the size of the character set, which in turn restricts the generalizability of the solutions. In addition, limited numbers of fitness cases can cause some degradation of the rule sets undergoing the upgrading process. For example, although the discussed successful individual in experiment 2 was perfect on the 10 002 fitness cases and all Eurostyle characters, changes made to accommodate the new Eurostyle characters affected generalization abilities on other fonts slightly (dropping from 99.999% to 99.95% on Courier). GP OCR is limited to the information contained in the training sets, whereas human programmers have a generalized notion of what makes a character a 'C' that has more inertia—learning a new font does not ruin previously learned information. Future work will examine the use of automatically defined functions (ADFs) (Koza 1994) to improve generalization. In addition, the effects of allowing individuals to use various forms of long-term individual and collective memory will be investigated.

One important facet of this research is that the language used in GP was designed to interact with and be equivalent to the language used by human programmers. The high number of functions and terminals used in this research was not for the mere sake of solving the problem, but for increased interaction with human-written programs. In fact, when genetic programming started from a random population, very few successful individuals used all of the functions or terminals. However, the large lexicon was required to allow translation from programs written in C. The possibility of easily combining evolved code and human-written code is exciting because it indicates that the different programming advantages of genetic programming and humans could perhaps be combined in a single system. This research not only suggests that genetic programming-evolved code can successfully interact with human-written programs, but also indicates that this approach for learning sets of rules, GP, is capable of learning complex rule sets for the difficult, noisy, real-world problem of OCR.

References

Amos L 1993 *A Method for Robust Segmentation of the Boundaries of Printed Characters* Canon Research Center Technical Report

Andre D 1993 *A Fast One Pass Raster-Scan Method for Boundary Extraction in Binary Images* Canon Research Center Technical Report

——1994a Automatically defined features: the simultaneous evolution of 2-dimensional feature detectors and an algorithm for using them *Advances in Genetic Programming* ed K Kinnear (Cambridge, MA: MIT Press) pp 477–94

——1994b Learning and upgrading rules for an OCR system using genetic programming *Proc. 1st IEEE Conf. on Evolutionary Computation (Orlando, FL, June 1994)* vol 1 (Piscataway, NJ: IEEE) pp 462–7

Andre D and Koza J R 1996 Parallel genetic programming: a scalable implementation using the transputer architecture *Advances in Genetic Programming* vol 2 ed P J Angeline and K E Kinnear (Cambridge, MA: MIT Press)

Handley S 1993 Automated learning of a detector for α-helices in protein sequences via genetic programming *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann)

Holland J H and Reitman J S 1978 Cognitive systems based on adaptive algorithms *Pattern Directed Inference Systems* ed D A Waterman and F Hayes-Roth (New York: Academic) pp 313–29

Koza J R 1992 *Genetic Programming: on the Programming of Computers by means of Natural Selection* (Cambridge, MA: MIT Press)

——1993 Simultaneous discovery of detectors and a way of using the detectors via genetic programming *1993 IEEE Int. Conf. on Neural Networks (San Francisco, CA, March–April 1993)* vol III (Piscataway, NJ: IEEE) pp 1794–801

——1994 *Genetic Programming II: Automatic Discovery of Reusable Programs* (Cambridge, MA: MIT Press)

Koza J R and Andre D 1996a Automatic discovery of protein motifs using genetic programming *Evolutionary Computation: Theory and Applications* ed Xin Yao (Singapore: World Scientific) in press

——1996b Classifying protein segments as transmembrane domains using architecture-altering operations in genetic programming *Advances in Genetic Programming II* ed P J Angeline and K E Kinnear Jr (Cambridge, MA: MIT Press)

Smith R E 1980 *A Learning System based on Genetic Adaptive Algorithms* Doctoral Dissertation, University of Pittsburgh

Tackett W A 1993 *Simple Genetic Programming Code* unpublished

——1994 *Recombination, Selection, and the Genetic Construction of Computer Programs* Doctoral Dissertation, University of Southern California, Department of Electrical Engineering Systems

Teller A and Veloso M 1995 PADO: a new learning architecture for object recognition *Symbolic Visual Learning* ed K Ikeuchi and M Veloso (New York: Oxford University Press)

Further reading

1. Angeline P J and Kinnear K E Jr (eds) 1996 *Advances in Genetic Programming* vol 2 (Cambridge, MA: MIT Press)

   A compilation of chapters presenting recent research in the field of genetic programming, including chapters on classification, recursion, iteration, new applications, evolution of program architecture, adaptive crossover, and parallelization of GP.

2. Kinnear K E Jr (ed) 1994 *Advances in Genetic Programming* (Cambridge, MA: MIT Press)

   A compilation of chapters presenting research in the field of GP, including chapters on GP theory, applications, classification problems, automatically defined functions, and memory use in GP.

3. Koza J R 1992 *Genetic Programming: On the Programming of Computers by means of Natural Selection* (Cambridge, MA: MIT Press)

   This is the seminal book on GP. It introduces the paradigm and convinces the reader by force of example that GP can solve a wide range of problems in many different domains.

4. Koza J R 1994 *Genetic Programming II: Automatic Discovery of Reusable Programs* (Cambridge, MA: MIT Press)

   This book describes the powerful technique of using automatically defined functions in GP and convinces the reader of their utility.

G8.2 Genetic programming applied to image discrimination

Walter Alden Tackett and K Govinda Char

Abstract

Automatic target recognition (ATR) involves the determination of objects in natural scenes in different weather conditions and in the presence of various contaminants. This high degree of variability requires a flexible system control capable of adapting to the changing conditions. There is no single set of adaptive algorithms that would give consistent, reliable results when subject to the full variety of target conditions. Although genetic programming (GP) has been successfully applied to a wide variety of problems its performance in scaling up to real-world situations needs to be addressed. In this case study we present the simulation results of applying GP to ATR through the development of a processing tree for classification of features extracted from images: measurements from a set of input nodes are weighted and combined through linear and nonlinear operations to form an output response. No constraints are placed upon size, shape, or order of processing within the network. This network is used to classify feature vectors extracted from infrared imagery into target/nontarget categories using a database of 2000 training samples. Performance is tested against a separate database of 7000 samples. This represents a significant scaling up from the problems to which GP has been applied to date. Two experiments are performed: in the first set, we input classical 'statistical' image features and minimize misclassification of target and nontarget samples. In the second set of experiments, GP is allowed to form its own feature set from primitive intensity measurements. For purposes of comparison, the same training and test sets are used to train two other adaptive classifier systems, the binary tree classifier and the multilayer perceptron/backpropagation neural network. The GP network achieves higher performance with reduced computational requirements. The contributions of GP 'building blocks', or subtrees, to the performance of generated trees are examined.

G8.2.1 Introduction

Genetic programming (GP; Koza 1992) has been demonstrated in a variety of applications, many of which have known optimal solutions determined in advance. This leaves open the question as to whether GP can 'scale up' to real-world situations, where answers are not known, data are noisy, and measurements may be of poor quality. We attempt to address this by constructing such a problem: noisy image data are segmented and processed into statistical features. These features are used to assign each image to a *target* or *nontarget* category. Because they are constrained by processing requirements, the segmentation and feature measurement are of a coarse and sometimes unreliable nature. Because of this it is likely that overlap between classes in feature space exists, and hence discovery of a 100% correct solution is unlikely. Two experiments are performed: in the first we insert a GP-generated tree into an existing system in order to test it against other well-established adaptive classifier technologies, specifically multilayer perceptron/backpropagation (Rumelhart and McClelland 1986) and decision tree (Quinlan 1983) methods. We then proceed to examine whether GP can be inserted at an earlier stage of processing, obviating costly segmentation and feature extraction stages.

B1.5.1

G8.2.2 The problem

G8.2.2.1 Target/nontarget discrimination

Target/nontarget discrimination is an important first stage in automatic target recognition (ATR), and in general for systems which require attention to a small subarea (or areas) within a much larger image. Whereas a highly sophisticated pattern recognizer may make fine discriminations between subtly different patterns (Daniell *et al* 1992), it generally does so at a very high computational cost. A much more efficient way to process information is to employ a simpler *detection* algorithm to the entire image, identifying subareas of interest. These areas are only coarsely classified as *target* or *nontarget*, and with lower reliability than the recognizer. Only target areas are passed from the detector to the recognizer, reducing image bandwidth and throughput requirements. Thus there is a constraint on this problem that GP must produce a solution which is computationally efficient in order to be useful. There is further advantage to be gained if GP is able to bypass costly processing associated with feature extraction.

G8.2.2.2 Image database

Data were taken from US Army NVEOD Terrain Board imagery. These are (512×640)-pixel images which simulate infrared views of vehicles, including tracked and wheeled vehicles, fixed- and rotary-wing aircraft, air defense units, and a wide variety of cluttered terrain. The range, field of view, and sensor resolution are such that individual targets occupy between 100 and 300 pixels. There are a total of about 1500 images containing multiple target and clutter objects, providing a total of about 13 000 samples to be classified. Training and test data for the experiments described here were drawn randomly from these samples.

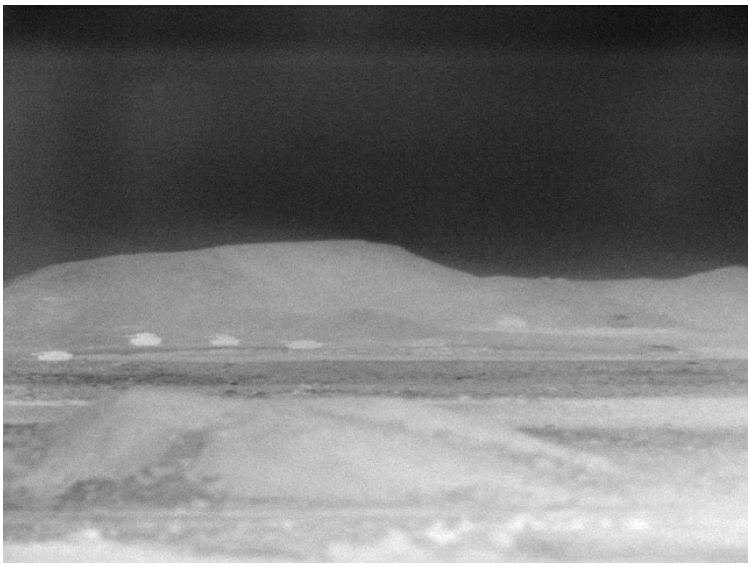


Figure G8.2.1. Eight (!) tank targets, light clutter (from US Army NVEOD Terrain Table database).

G8.2.2.3 Feature extraction

In the experiments described here we have used GP to construct classifiers which process the feature vectors produced by an existing algorithm—specifically the multifunction target acquisition processor (MTAP) ATR system (Hughes Aircraft Co. 1990). This system performs two sequential steps of image processing.

Antimean detection filter. This system uses an antimean filtering to extract *blobs* in a range of sizes conforming to those of expected targets. The image is divided into a mosaic of (62×62)-pixel overlapping regions, or *large windows*. These regions are themselves divided into overlapping 5×5 *small windows*.

When a blob of appropriate size is detected, it is described in terms of seven primitive features listed in table G8.2.1: contrast with the local background, global image intensity mean and standard deviation, and the means and standard deviations of the *large window* and *small window* in which it is centered. Later we will apply GP to the classification of these features. In the conventional MTAP algorithm, however, they are passed to a second processing stage.

Table G8.2.1. Seven primitive features.

F00	Size filter value ('blob contrast')
F01	Global image intensity mean
F02	Global image intensity standard deviation
F03	'Large window' intensity mean
F04	'Large window' intensity standard deviation
F05	'Small window' intensity mean
F06	'Small window' intensity standard deviation

Segmentation and feature extraction. In the MTAP system, the seven features from the antimean detection filter are passed through a simple linear classifier in order to determine whether segmentation and feature extraction should be performed upon the blob. If so, the large image window undergoes a 3:1 decimation filtering to reduce bandwidth and statistical measures of local intensity and contrast are used to perform a binary figure/ground segmentation, resulting in a closed-boundary silhouette. Twenty moment- and intensity-based features are extracted from this segmented region. They are summarized in table G8.2.2. Because this segmentation scheme depends on fixed thresholds under varying image conditions, silhouettes for a given target type can vary significantly. Likewise, since features do not encode detailed shape properties it is possible for nontarget objects such as oblong rocks to be encoded into target-like regions of feature space. Thus these feature vectors may display significant variance as well as overlap between target and nontarget classes.

Table G8.2.2. Twenty statistical features from the segmented image.

F00	radius of gyration
F01	rectangularity
F02	$\text{height}^2/\text{width}^2$
F03	$\text{width}^2/\text{height}^2$
F04	normalized contrast
F05	symmetry
F06	range to target
F07	depression angle
F08	$\text{perimeter}^2/\text{area}$
F09	normalized average segmentation graylevel
F10	area
F11	$\text{height}^2/\text{area}$
F12	$\text{height}^2/\text{range}^2$
F13–F17	higher-order moments
F18	$\text{area}/\text{range}^2$
F19	polarity of contrast

G8.2.3 Approach

The fitness of an individual tree is computed by using it to classify about 2000 *in-sample* feature vectors. At each generation, the individual which performs best against the training set is additionally run against 7000 out-of-sample feature vectors which form a validation test set. Only results against the *out-of-sample* validation set are reported.

G8.2.3.1 The function set

Both experiments share a common function set: $F = \{+, -, *, \%, \text{IFLTE}\}$ represents four arithmetic operations which form second-order nodes (i.e. two arguments) and a conditional operation which forms fourth-order nodes (four arguments). The $+$, $-$, and $*$ operators represent common addition, subtraction, and multiplication, while % indicates 'protected' division: division by zero yields a zero result without error (Koza 1992). The conditional IFLTE returns the value of the third argument if the first argument is less than the second, otherwise the fourth argument is returned.

G8.2.3.2 The terminal set and fitness function for experiment 1

The terminal set $T = \{\text{F00}, \ldots, \text{F19}, \text{RANFLOAT}\}$ for the first experiment consists of the 20 segmented 'statistical' features shown in table G8.2.2 as well a real random variable RANFLOAT, which is resampled to produce a constant value each time it is selected as a terminal node. The resulting tree takes a bag of floating-point feature values and constants as input, and combines them through linear and nonlinear operations to produce a numeric result at the root of the tree. If this result is greater than or equal to zero, the sample is classified as a target, otherwise it is classified as clutter (nontarget). Testing of trees against the 2000-sample set results in two measurements: the first is probability of incorrect classification, or the total fraction of samples assigned to the incorrect category. It had previously been observed that performance could be enhanced by including a larger number of target samples than clutter samples in the training database (see section G8.2.5). This gives rise to the problem that a classifier which says *everything* is a target may achieve a relatively high fitness score while conveying no information in the Shannon sense (Shannon 1948). To counteract this, a second measure of fitness was added: the *a posteriori* entropy of class distributions $H(\text{class}|\text{output})$ after observing classifier output. These multiple objectives are minimized via mapping to the *Pareto plane* and subjecting them to a nondominated sort (Goldberg 1989a). The resulting ranking forms the raw fitness measure for the individual. C4.5.3.7

G8.2.3.3 The terminal set and fitness function for experiment 2

In the second experiment, only the seven primitive intensity features from the antimean discriminant filter are used. The terminal set $T = \{\text{F00}, \ldots, \text{F06}, \text{RANINT}\}$ consists of these seven integer features and an integer random variable which is sampled to produce constant terminal nodes.

The fitness function is reformulated from experiment 1 for 'historical reasons', namely the constraints that were placed on the original MTAP system. These state that the detection filter should be able to find five targets in an image with 80% probability of detecting all of them. This means that individual probability of detection ($p(\text{D})$) must be $(0.8)^{0.2}$, or about 96%. With this figure fixed, we seek to minimize the probability of false alarms ($p(\text{FA})$), the chance that nontargets are classed as targets. This is done in the following manner: an expression is tested against each sample in the database, and the values it produces are stored into two separate arrays, one for targets and the other for clutter. These arrays are then sorted and the threshold value is found for which exactly 96% of the target samples produced a greater output. We compare this value with those in the clutter array in order to determine what percentage fall above this threshold value. This percentage is the false alarm rate which we seek to minimize.

G8.2.4 Performance comparison

G8.2.4.1 Multilayer perceptron and binary tree classifier

As an experimental control the same test and training databases are used to build and evaluate two other classifiers. The first is a multilayer nonlinear perceptron *neural network* trained via the backpropagation algorithm with adaptive step size η (Hertz *et al* 1991) and two output nodes (one for each class). For both experiments, the 'desired output' activation for this neural net is $\{1.0, 0.0\}$ when presented with a target sample, and $\{0.0, 1.0\}$ when presented with a clutter sample. Differences between these desired outputs and those actually obtained form error terms to be fed back. Weight updates take place after each sample presentation. Training inputs to the network are normalized so that values do not exceed an absolute value of 1.0. A threshold difference between network outputs is used to determine whether a sample is target or clutter. This threshold is 0.0 for experiment 1 and variable for experiment 2. The second classifier tested is a binary decision tree classifier (Quinlan 1983, Kanal 1979) which partitions

feature space into hyperrectangular regions, choosing features and their thresholds at decision nodes based upon the maximization of mutual information (also known as information rate) (Popoulis 1965:1984). This particular binary tree classifier was originally developed and optimized for use with the feature set of experiment 1 (Hughes Aircraft Co. 1990). It was not used in experiment 2.

G8.2.4.2 A basis for comparison

In performing comparisons between methods there is some question as to what is a fair basis for assessing the work which is required to obtain results for each. When this effort was started central processing unit (CPU) time was not a fair basis since the original GP system was written in LISP, which is less computationally efficient than the C language used to implement this version of backpropagation. Instead, we consider that there was equivalent human effort required to make the two methods work correctly. Both the GP and neural net systems were obtained from second-hand sources. GP was run using a standard function set which was included in the software and had previously been used in the *Nested Spirals* classification problem (Koza 1992). All of the GP system parameters were left at their default settings as were those of the backpropagation algorithm. In both cases, the default parameter settings had previously shown satisfactory performance on several other problems. Some cursory experimentation was performed with GP to determine a minimum population size which produced adequate performance, and likewise some tests were run with the neural network to determine the appropriate numbers of hidden layers and neurons. A few hours of labor were required to code and debug the fitness functions used for GP in experiments 1 and 2. The perceptron network required normalization of input data, which was not required by GP. Results for the decision tree used in experiment 1 were obtained previously under a separate study (Hughes Aircraft Co. 1990).

G8.2.5 Results

For all experiments performance was plotted as a function of probability of false alarm $p(\mathrm{FA})$ against probability of detection $p(\mathrm{D})$. Probability of false alarm is the fraction of nontargets classified as targets divided by the total number of nontarget samples. Probability of detection is the fraction of targets correctly classified divided by the total number of target samples. Ideally a system should achieve a $p(\mathrm{FA})$ of 0.0 and a $p(\mathrm{D})$ of 1.0.

G8.2.5.1 Experiment 1

A total of twelve runs were performed using GP with different random seeds. Three different cluster-to-target ratios (*C/T* ratios) were used in the training database: 0.5:1, 0.71:1, and 1:1, with 0.5:1 consistently producing the best results. Each run used a population of 500 and run length of 60 generations, resulting in the analysis of a total of 120 000 individuals for each *C/T* ratio. Backpropagation was similarly tested using three *C/T* ratios, again with 0.5:1 producing the best results. The network used 20 input nodes (the 20 features) and ten hidden nodes (empirically determined to be the best number). Four random initial weight sets were used for each *C/T* ratio, resulting in a total of 12 learned weight sets. After each training epoch each network was tested against the separate validation test set. The best result achieved against this test set was reported as the figure of merit.

The binary tree classifier was tested using the three *C/T* ratios, and additionally tested with an input set that eliminates gray-level features. There is no random initial element. Various system parameters of the tree builder were refined during its development cycle.

Figure G8.2.2 summarizes the performance of the three classifiers. The three points corresponding to backpropagation depict the best performance achieved against the three *C/T* ratios. The six points shown for the binary tree correspond to three *C/T* ratios for the two input feature sets described above (the cluster of three points with higher $p(\mathrm{FA})$ was obtained with gray-level features omitted). The ten points shown for GP are the best-of-generation individuals for selected generations in a single evolution run with *C/T* ratio of 0.5. All three methods were able to achieve $p(\mathrm{D})$ of 74–75%. Genetic programming, however, achieves a 31% false alarm rate, some 8% lower than the 39% $p(\mathrm{FA})$ achieved by backpropagation for the same $p(\mathrm{D})$, and 30% lower than the 61% figure achieved by the binary tree.

The best-of-run individual for generation 48 (indicated by the arrow in figure G8.2.2) of the 'most successful' GP run for experiment 1 is shown below, with the dendritic tree represented

in LISP program format. This program achieved 74.2% correct with a 30.8% false alarm rate. Counting function nodes we see that 55 mathematical operations required and there are 15 logical comparisons/branches. By comparison, the backpropagation network requires 440 math operations and 12 nonlinearities: about 8 times more computation is required in order to achieve 8% *lower* performance.

```
(+
  (*F07
    (IFLTE
      (*F07 F04)
      (-
        (-
          (*
            (IFLTE
              (*F13 F06)
              (-F16 F03)
              (IFLTE F17 F19 F14 F07)
              (+F09 F09))
            (+F06 F01))
          (%F02 F00))
        F11)
      (-F06 F04)
      (+-3.335 647 232 986 822 5 F04)))
  (+
    (-
      (+
        (IFLTE
          (*F13 F06)
          (+-3.335 647 232 986 822 5 F04)
          (+(+-3.335 647 232 986 822 5 F04)
            (%F02 F00))
          (+F09 F09))
        F04)
      F11)
    (-
      (-
        (IFLTE
          (IFLTE F16 F16 F19(-F04 F11))
          (*(*F13 F06)(+F06 F01))
          (IFLTE F16 F15 F04 F01)
          (*F07
            (IFLTE
              (*F07 F04)
              (+-3.335 647 232 986 822 5 F04)
              (%F11 F11)
              (+F04 F02))))
(+
  (*
    (-
      (IFLTE F16 F16 F19(+F09 F09))
      (*F07 F04))
    (+F09 F09))
  (-
    (*(-F01 F07)
      (IFLTE F14 F08 F06 F08))
    (+(-F16 F03)
      (*
        (IFLTE
          (+-3.335 647 232 986 822 5 F04)
          (IFLTE
            F14
            -3.335 647 232 986 822 5
            F12
            F03)
          (IFLTE
            (*F13 F06)
            (IFLTE
              F16 F16 F19
              (-F04 F11))
            F02
            (-F06 F04))
          (+F04 F02))
        (IFLTE
          (*F13 F06)
          (-F16 F03)
          (+-3.335 647 232 986 822 5 F04)
          (+F09 F09)))))))
(*F10 F08))))
```

G8.2.5.2 Experiment 2

As with experiment 1, GP was run four times for 60 generations with a population of 500 and three *C/T* ratios using the fitness function described in section G8.2.3.3. Training and testing for the backpropagation network were performed exactly as described in sections G8.2.4 and G8.2.5.1, except that this network contained seven input nodes and four hidden nodes. For both GP trees and neural networks a *C/T* ratio of 0.5 : 1 resulted in best performance.

Figure G8.2.3 depicts the performance of the backpropagation and GP systems. The dashed vertical

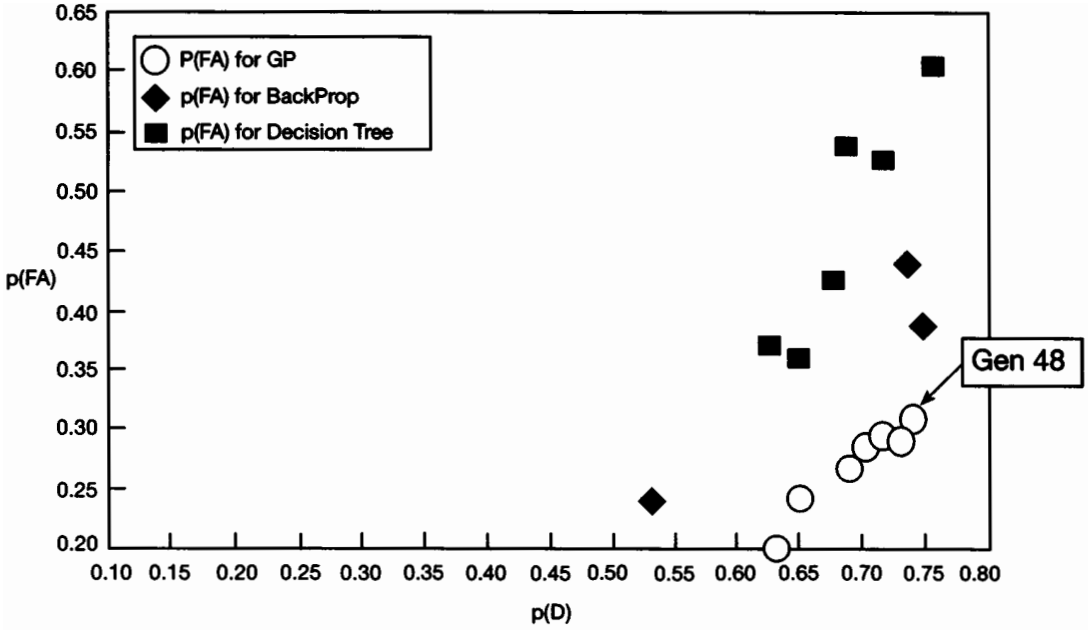


Figure G8.2.2. The comparative performance of GP, neural, and decision tree methods. The objective is to minimize the probability of false alarm $p(\mathrm{FA})$ while maximizing the probability of detection $p(\mathrm{D})$.

line indicates the desired $p(\mathrm{D})$ of 96%. For this value of $p(\mathrm{D})$ the best expression derived by GP produced a 65.8% false alarm rate, while the best network derived by backpropagation produced 82.6%. By varying the $p(\mathrm{D})$ threshold, we generate curves showing $p(\mathrm{FA})$ for these other values. This provides an insight about the two algorithms: GP used a fitness function which was specifically tuned to the desired $p(\mathrm{FA})$ performance, whereas the backpropagation training was independent of the threshold (used only *after* training) with no straightforward way to incorporate the 96% constraint. Thus GP chooses a function which specifically trades off performance at high $p(\mathrm{D})$ values for performances at lower $p(\mathrm{D})$ values.

The individual whose performance is depicted in figure G8.2.3 is shown in the following LISP expression form (best-of-generation 5 for the 'most successful' GP run of experiment 2):

```
(%(−F02 F00)
  (+(+(%(−(∗F02 F03)F06)
           (+(−F02 F00)(−F03 F00)))
       (−F03 F00))
    (−F03 F00))
```

This function requires 12 mathematical operations, compared to the 72 math operations and six nonlinear function evaluations required by the backpropagation network. In and of itself it displays some remarkable properties: it does not make use of the IFLTE conditional branch, and ignores the input features F01, F04, and F05. Likewise, the system makes repeated use of synthesized 'metafeatures', (−F02 F00) and (−F03 F00). Clearly this gives insight into which features are and are not useful for the low-level discrimination task. Another interesting interpretation of these results is that F03 and F02 are localized measures of intensity, while F00 is a global measure of contrast. In terms of image processing, the expressions (−F02 F00) and (−F03 F00) represent local measures of contrast which are the kind of features a human expert might formulate: indeed, measures of contrast form some of the 20 features used in experiment 1! In theory, this solution could tell a user with little expertise in image processing that contrast is a powerful feature.

The atypically small size of this expression allows us to make an analysis of its major subcomponents in terms of their individual fitnesses: figure G8.2.4 shows performance obtained by decomposing this expression into its three unique subtree components, which we postulate comprise *GP building blocks* (Holland 1992). The building block (- F02 F00) appears, at $p(\mathrm{D})$ values below 60%, to display performance in the same ballpark as that of the backpropagation network. Significantly, all three schemata approach

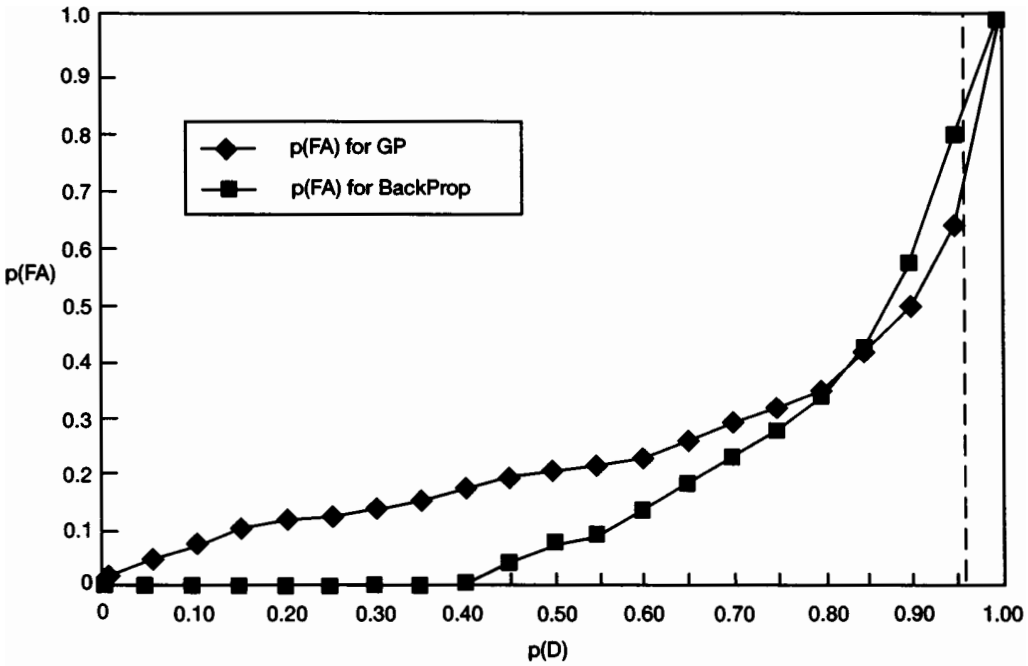


Figure G8.2.3. A tradeoff between detection and false alarms can be achieved by varying the threshold which distinguishes targets from nontargets. The intent of experiment 2 is to minimize false alarms at the point $p(\mathrm{D}) = 0.96$ using a set of 'primitive' intensity measurements.

100% $p(\mathrm{FA})$ at 96% $p(\mathrm{D})$. Thus they are contributing to the fitness of the parent expression in no way by a principle of superposition, but rather by their nonlinear interactions. Although GP may have a different notion of schemata and a nonbinary alphabet, we suggest that this observation underscores principles previously set forth by Goldberg (1989b).

G8.2.6 Discussion

G8.2.6.1 Bias sources and statistical validity of experiments

It is usually not too hard to show one's favorite method to be better than others. For this reason we have gone to some length to describe the equivalence of effort put into the methods described (see section G8.2.4.2). In the interest of fair comparison of results some further points must be discussed.

First, for both experiments the GP method uses fitness functions more sophisticated than the simple mean-squared-error minimization inherent to the backpropagation algorithm. Indeed, the extra effort required to do so constitutes the only source of special 'hand tuning' that went into either method. This cannot be avoided, since GP has no inherent measures of error or fitness: one must always create a mapping from algorithm performance ('phenotype') to a scalar fitness measure. This may be viewed as an asset when one considers that specialized measures of performance such as those used here for GP cannot be easily expressed in terms of the generalized delta rule used for backpropagation.

In order to achieve a reasonable data generation and reduction task for the given time and resources, only twelve runs each were made for GP and backpropagation. Of these, only four were performed at each *C/T* ratio. It would be desirable to create a large number of runs at optimal *C/T* values with different random seeds in order to obtain statistically significant mean and standard deviation values for performance. Despite the lack of such extensive statistics, the fact that for two separate classification tasks GP achieved a significant performance increase and greatly reduced computational complexity is a forceful argument in favor of the power of *structural adaptation*. Moreover, recent work (Carmi 1994) has put extensive effort into training this same neural network system against the induction problem and the advantage of the GP method is even greater in that problem than in the results shown here.

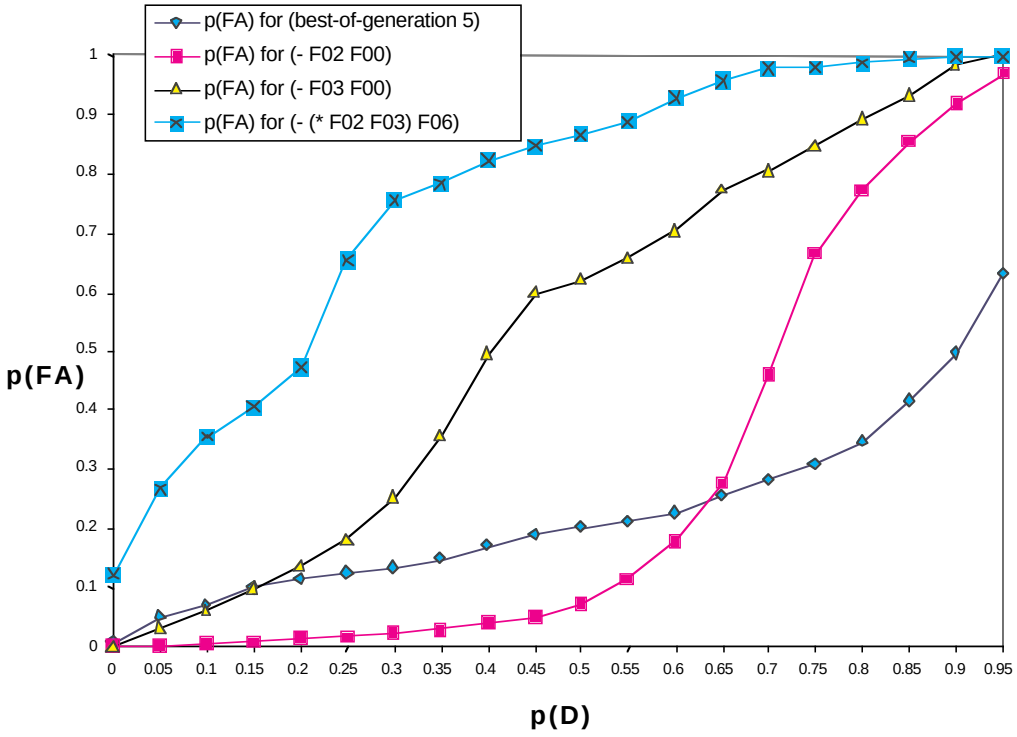


Figure G8.2.4. Performance of individual subexpressions and the program which contains them.

G8.2.6.2 Primitive features and improved performance

One significant observation is that the performance obtained in experiment 2 is actually better than that of experiment 1: for a p(D) of 75%, the GP tree of experiment 2 obtained 30% p(FA) while the backpropagation net obtained 27%. This is unlikely to be due to coincidence since both methods showed improved performance. Why else, then? One hypothesis is that both of these powerful nonlinear adaptive methods may be discovering inherent features which are better suited to the data than human-synthesized features based upon measurements (incorrectly) presupposed to be invariant. We may also speculate that the segmentation process introduces artifacts and variations which do not exist in the raw data. Other factors must be considered: we have mentioned that the image resolution is reduced prior to feature extraction, compressing groups of 3×3 (i.e. nine) pixels into one. This reduction in bandwidth may be harmful. Finally, the reduction in the number of input features itself may exponentially reduce the complexity of problem space to be searched, making relatively good solutions easier to find. Regardless of the cause we may conclude that this appears to be a result of direct benefit to the development of future systems. It also suggests that applying GP to raw pixel imagery is a promising area for follow-on research.

G8.2.6.3 Genetic programming offers problem insights

We have seen that in the case of experiment 2, the structure of the solution tree offers clues as to which features assist in pattern discrimination. Although it is more complex, the same analysis can in principle be applied to experiment 1. This indicates that GP can provide insights into what aspects of the problem are important: we can understand the problem better by examining the solutions that GP discovers.

G8.2.6.4 A genetic programming schema theory?

Upon examination, we see that there are many repeated structures in the tree of experiment 1 as well as that of experiment 2. By probability, these are *not* identical structures randomly generated at the outset, but rather successful subtrees, or *building blocks*, which were frequently duplicated and spread throughout the population at an exponential rate due to their contribution to the fitness of individuals containing them.

These suggest that a schema theory may be developed for genetic programming analogous to that for binary GA originally proposed by Holland (1992).

G8.2.6.5 Parsimony

Parsimony, or simplicity of solution structures, has previously been shown by Koza (1992) to be achievable by adding the tree size (expression length) as a component of the fitness function to be minimized. In both experiments it was observed that a high degree of correlation exists between tree size and performance: among the set of 'pretty good' solution trees, those with highest performance were usually the smallest within that set. It was also observed that most runs achieved a point where the size and complexity of trees eventually began to grow increasingly larger, while performance tapered off to lower values. We suggest that, for an open-ended exploration of problem space, parsimony may be an important factor, not for 'esthetic' reasons or ease of analysis, but because of a more direct relationship to fitness: there is a bound on the 'appropriate size' of the solution tree for a given problem.

G8.2.6.6 Learning, central processing unit time, and LISP

There is a significantly higher computational cost for the generation of GP solutions relative to that of the multilayer perceptron architecture. A training run (one random seed, one *C/T* ratio) using the backpropagation algorithm requires about 20 minutes of CPU time on a Sun SparcStation/2. By comparison, an equivalent training run for GP using compiled LISP can take 40–50 hours of CPU time. This clearly has an impact on the ability to generate large or complex GP runs. Recently, we have created a new C language implementation, GP/C (Genetic Programming in C). This code represents programs internally as parse trees in tokenized form. For backwards compatibility the trees may be read and written as LISP expressions. For the problem described in experiment 2 we have compared GP/C with equivalent LISP code. A typical run for GP/C requires 2 hours of CPU time for 60 generations and a population of 500, about a 25-fold improvement in throughput relative to the LISP implementation. The 2 hours of CPU required by GP/C is still about six times greater than is required by the backpropagation algorithm for the same problem. This provides an interesting tradeoff of training against execution time, given the more efficient runtime performance achievable by the GP-generated solutions.

References

Carmi A 1994 *The donut problem II: a comparative performance of genetic programming and neural networks* Master's Thesis, Department of Computer Science, California State University at Northridge

Daniell C E, Kemsley D H, Lincoln W P, Tackett W A and Baraghimian G A 1992 Artificial neural networks for automatic target recognition *Opt. Eng.* **31** 2521–31

Goldberg D E 1989a *Genetic Algorithm in Search, Optimization, and Machine Learning* (Reading MA: Addison Wesley)

——1989b Zen and the art of genetic algorithms *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, June 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann)

Hertz J, Krogh A and Palmer R G 1991 *Introduction to the Theory of Neural Computation* (Redwood City, CA: Addison-Wesley)

Holland J H 1992 Adaptation in natural systems and artificial systems (Cambridge, MA: MIT Press)

Hughes Aircraft Co. 1990 *Bandwidth Reduction Intelligent Target-Tracking/Multi-function Target Acquisition Processor (BRITT/ MTAP)* Final Technical Report, CDRL B0001

Kanal L N 1979 Problem-solving models and search strategies for pattern recognition *IEEE Trans. Pattern Anal. Machine Intell.* **PAMI-1** 194–201

Koza J R 1992 *Genetic Programming* (Cambridge, MA: MIT Press)

Popoulis A 1965:1984 *Probability, Random Variables, and Stochastic Processes* (New York: McGraw-Hill)

Quinlan J R 1983 Learning efficient classification procedures and their application to chess end games *Machine Learning* ed R Michalski, J Carbonell and T Mitchell (Palo Alto, CA: Tioga) ch 15

Rumelhart D E, and McClelland J 1986 Parallel distributed processing vol 1 (Cambridge: MIT Press)

Shannon C E 1948 A mathematical theory of communications *Bell Syst. Tech. J.* Part I: 379–423; Part II: 623–56

G8.3 Tracking a criminal suspect through face space with a genetic algorithm

Victor S Johnston and Craig Caldwell

Abstract

Although humans have excellent facial recognition ability, they often have great difficulty recalling facial characteristics in sufficient detail to generate an accurate composite of a criminal. As a consequence, facial composite systems that depend on human recall are not always adequate. FacePrints is an alternative approach that uses recognition to guide a genetic algorithm search of a *face space* containing over a billion billion possible facial composites. Using subjective estimates of resemblance to a culprit as a measure of fitness, a witness can interactively exert selection pressure on the search process and evolve the likeness of a criminal suspect over a small number of generations. FacePrints performs the double function of generating a facial composite and a binary code for that composite. These binary strings can serve as codes for individual faces, not unlike fingerprints, and they may provide a convenient method for searching a database of known criminals to find those that most closely resemble a generated composite. Methods for increasing the efficiency of the search procedure are examined. Results indicate that a genetic-algorithm-based search is significantly superior to the standard method for generating facial composites.

G8.3.1 Introduction

Human users can interact with a *genetic algorithm* in real time. When engaged in a multidimensional search problem, a user's preferences can provide the measure of fitness needed to steer the algorithm toward a desired goal. When fitness is based on the user's esthetic preferences, the search process can converge on a an esthetically pleasing outcome, such as a piece of music or a beautiful face (Johnston and Franklin 1993). If the fitness of phenotypes is based on a cognitive skill, such as recognition, the algorithm can evolve a solution using this ability. One area where an interactive genetic algorithm has been useful is in assisting a witness to build a facial composite of a criminal suspect (Caldwell and Johnston 1991). B1.2

Humans are experts in facial recognition. They can recognize and discriminate between a very large number of faces seen over a lifetime, often following a single short exposure. In contrast, humans have poor recall ability; they may not be able to recall the features of a close associate, or even a family member, in sufficient detail to construct a facial composite (Ellis *et al* 1986, Goldstein and Chance 1981, Rakover and Cahlon 1989). As a consequence, current facial composite procedures, which depend heavily on recall rather than recognition, may not be using the best approach for generating an accurate composite of a target face.

One of the most widely used systems for generating composite faces was developed by Penry (1974), in Britain, between 1968 and 1974. Termed *Photofit*, this technique uses over 600 interchangeable photographs, picturing five basic features: forehead and hair, eyes and eyebrows, mouth and lips, nose, and chin and cheeks. With additional accessories, such as beards and eyeglasses, combinations can produce approximately fifteen billion different faces. Alternatives to Photofit include the Multiple Image-Maker and Identification Compositor (MIMIC), which uses film strip projections, Identikit, which uses plastic

overlays of drawn features, and several computerized versions of the Photofit process, such as Mac-A-Mug Pro and Compusketch. Using Compusketch, a trained operator with no artistic ability can assemble a composite in less than an hour. Because of such advantages, computer-aided sketching is becoming the method of choice for law enforcement agencies.

Facial identification involves gestalt processes (Homa *et al* 1976, Perkins 1975), and varies with the age, sex, cognitive style, and hemispheric advantage of the witness (Going and Reed 1974, Hall 1976, Ellis *et al* 1978, Yarmey and Kent 1980, Goldstein and Chance 1981, Solso and McCarthy 1981, Miller and Barg 1983, Ross-Kossak and Turkewitz 1986, Hines *et al* 1987). Systems such as Photofit and Compusketch depend on the ability of a witness to accurately recall the features of a suspect and to be aware of which features and feature positions of the generated composite require modification. Such systems may actually inhibit identification by forcing a witness to employ a specific cognitive strategy, namely, the recall of isolated features. Davies and Christie (1982) have shown that this single-feature approach is a serious source of distortion, and Baddeley (1979) has concluded that any exclusively feature-based approach is misconceived.

G8.3.2 Design process

An alternative approach, *FacePrints*, utilizes a genetic algorithm and relies on recognition rather than recall (Johnston 1994). Unlike other contemporary procedures, FacePrints is capable of efficiently searching a large sample space of alternative faces and finding an accurate likeness to a culprit in a relatively short period of time. FacePrints dynamically interacts with a witness. No artistic ability is required of a witness, and no biasing influences are introduced by interacting with the program. Additionally, the FacePrints implementation makes no assumptions concerning the age, gender, hemispheric advantage, or cognitive style of a witness, and can find an adequate solution irrespective of these influences (Caldwell and Johnston 1991).

The production of a culprit's face can be regarded as a search for a unique point in a multidimensional *face space* where the dimensions of the space correspond to the shape and position of facial features. The features and proportions of the target, as well as any other face in face space, can be expressed as a set of coordinates. The target face is a particular point, with unique values on the hair, eye, mouth, nose, ear, and chin axes, as well as unique proportions specified by coordinates on six position axes. For any face, this sequential set of coordinates can be coded as a *binary string* of length N. The search for the culprit's face can then be viewed as a search for a unique face out of 2^N possibilities. C1.2

In its current configuration the FacePrints program begins by generating a set of 30 random binary number strings (genotypes) and developing these into composite faces (phenotypes). During development, the features of each face are selected from a database and positioned on the computer screen according to the coordinate values coded as its genotype. With a string length of 60 bits, combinations of features and positions allow over 1.1 billion billion (2^{60}) different facial composites to be generated. The process of decoding the binary string, selecting the gray-scale features, and generating a composite, requires less than 1 second to compute.

'Selection of the fittest' from this first generation of random faces is the first step in the procedure. A witness sequentially views all 30, first-generation composites, and rates each face on a ten-point scale according to its resemblance to the culprit. Fitness ratings are entered using the '0' through '9' keys on a computer keyboard. These ratings need not depend upon the identification of any specific features shared by the culprit and the composite face; indeed the witness may not be aware of why any perceived resemblance exists. After fitness ratings are collected for the first generation of faces, the genotype of the fittest face and a second genotype, chosen in proportion to fitness from the remaining 29 faces, are paired for breeding. *Selection* in proportion to relative fitness is achieved using a stochastic universal sampling procedure (Baker 1987). C2.2

The next step in the process is the mating of the selected pair of genotypes. Breeding employs two operators: *mutation* and *crossover* (Spears and De Jong 1991). When two genotypes mate, they exchange random portions of their binary strings according to a prespecified crossover rate, and mutate according to a prespecified mutation rate (described below). Following selection and breeding with crossover and mutation, the two offspring genotypes are developed into faces, displayed individually on the computer screen, and rated by the witness for their resemblance to the target face. If either face is rated more highly than the least-fit face in the current population, then the genotype of the latter is discarded (dies) and is replaced by the new genotype (offspring). The population size remains constant as its average fitness C3.2, C3.3

increases over generations. The processes of selection and breeding continue until the witness concludes that a satisfactory composite of the culprit has been evolved.

The genetic algorithm provides a nominal search strategy that finds the most 'fit' outcome from a choice of evolutionary paths through the facial hyperspace. Evolutionary progress can be represented conceptually as a track through face space, from a random initial location to a 'region of recognition' encompassing the target point. Progress appears as a stochastic walk, connecting the most fit points of the evolved offspring. This stylized progression through face space is illustrated in figure G8.3.1. The strength of the algorithm lies in its simultaneous search along all of the dimensions of the problem, the exponential increase over generations of any partial solution that is above average fitness, and the exploration of small variations around these partial solutions (Holland 1975, Goldberg 1989). The path's convergence is enhanced by the sharing of partial solutions, through crossover, and the exploration of variations around these partial solutions, using mutations. The result is an effective search process that can evolve a match to a target face from over 1.1 billion billion possibilities, in less than 1 hour and fewer than 200 offspring. By contrast, a sequential search of the same face space, viewing one face every second, would require more than 36 billion years!

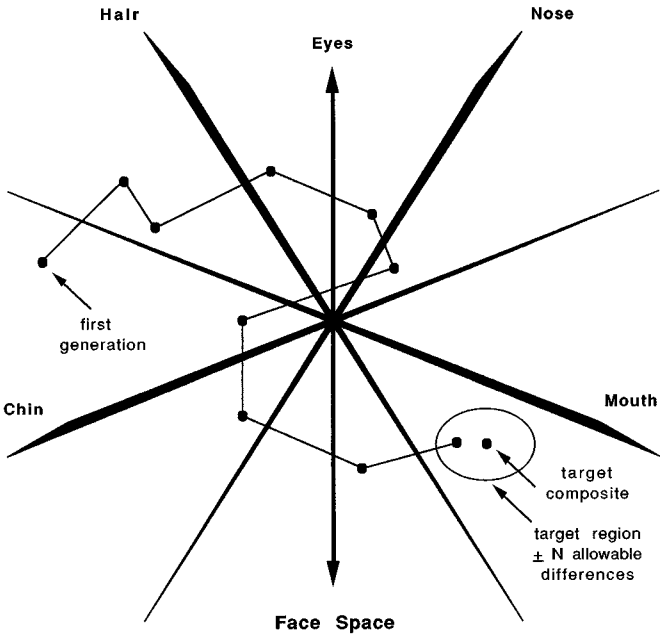


Figure G8.3.1. Evolutionary progress over generations may be conceptualized as a path through face space that approaches a region of faces resembling a target face. A point represents the most fit face in each generation.

G8.3.3 Development and implementation

In developing the FacePrints program, several procedures were implemented to accelerate the evolutionary progress toward the region of the target face in face space. Improving the genetic-algorithm-based search can be equated with streamlining this movement.

Suh and Gucht (1987) have examined how knowledge can be incorporated into a genetic algorithm to increase the efficiency of the search. One approach is to optimize the starting location of the search. Normally, the initial set of coordinates is randomly generated. However, if a witness has a clear memory of any of the culprit's features, this recollection can be exploited to constrain the location and variance of the first-generation genotypes to the most favorable areas of face space. Divisions by gender, skin complexion, hair length, and so on, can be used to reduce the scope of any feature dimension. A rapid search of these narrower regions of each database may be used to produce a crude composite that permits the witness to adjust facial proportions in context. From this point, the first generation of faces can be generated as random variations around this starting location, using the subject's certainty to define the explored variance along any feature dimension. This constraint procedure is an optional part of the FacePrints process.

The search may be further facilitated by permitting a witness to *freeze* a desirable feature of a displayed composite. Such freezing is implemented by overwriting the corresponding gene segments of the entire population of bitstrings with the appropriate code, in all future offspring. This immediately confines all future search to the remaining dimensions of face space. An alternative to freezing is to *flood* the population with the feature code, but permit mutations to subsequently alter the sequence. A simulation program was designed to act as a 'perfect' witness that could accurately rate each composite according to its exact phenotypic distance from the target in face space. Results from experiments with the program indicate that both the flood and freeze options produce a marked improvement in the performance of the algorithm (Caldwell and Johnston 1991). However, as composites begin to converge to the likeness of the culprit, mutations that previously increased the scope of the search now have a higher probability of being disruptive. Consequently, freezing is superior to flooding. This is illustrated in the performance plotted in figure G8.3.2. The freeze option has been included in the implementation of the FacePrints program to increase the efficiency of the search.

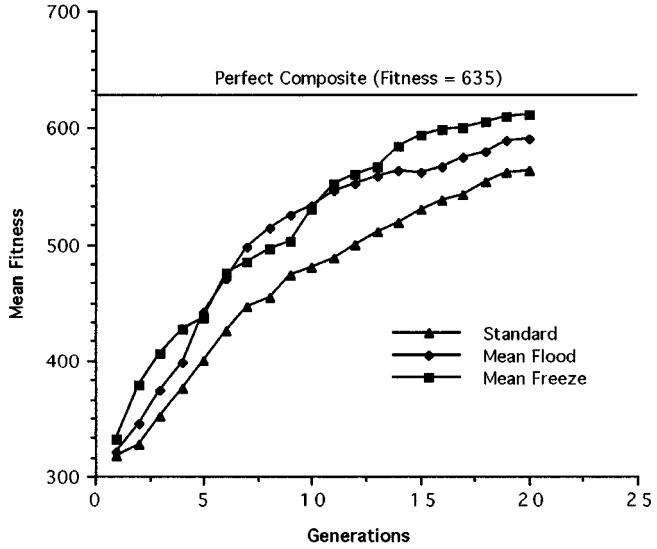


Figure G8.3.2. A comparison of the effects of the flood and freeze processes. Each generation in the simulation has a population of 20 genotype strings, each encoding a phenotypic composite face.

Different genotypic coding systems can also influence the rate of convergence to a known target. In binary, a change from number 011 (base ten: 3, gray code: 110) to number 100 (base ten: 4, gray code: 010), requires three simultaneous bit mutations: a Hamming distance of three. Some research indicates that gray code may facilitate search since there is a constant Hamming distance of one between adjacent values (Bethke 1981, Caruana and Schaffer 1988). However, convergence tests, using binary, gray, and bingray (in bingray the most significant bits are decoded as binary code and the least significant bits are decoded as gray code), indicate a superiority for binary code under the current search conditions. The rationale for examining bingray code is that altering the binary segment allows rapid transitions to more interesting regions of face space, while the gray segment changes permit smooth refinements in a local search. A curve showing the performance of all three codes is presented in figure G8.3.3. Over multiple simulations, the performance at generation 20 using binary (88.6%) has proved superior to both gray (81.1%) and bingray (87.1%). The problem with gray appears to arise from an inconsistent base ten value associated with any bit position. A statistical analysis of the data, using the method of contrasts, also shows that a binary-coded genetic algorithm converges significantly faster. FacePrints employs a binary code.

The crossover rate determines how rapidly the probability density distribution of points narrows around sequential segments of the walk through face space. For a string of length N, when only one crossover occurs at a random location, only $N-1$ strings may be formed from the original, or $2(N-1)$ for both breeding strings. When multiple crossovers are allowed, with an equal probability at each location, 2^N-1 reconfigurations are possible. This parameterized uniform crossover permits a wider and more thorough search, augmenting mutation effects by also allowing a single bit crossing to occur (Syswerda 1989, Spears and De Jong 1991).

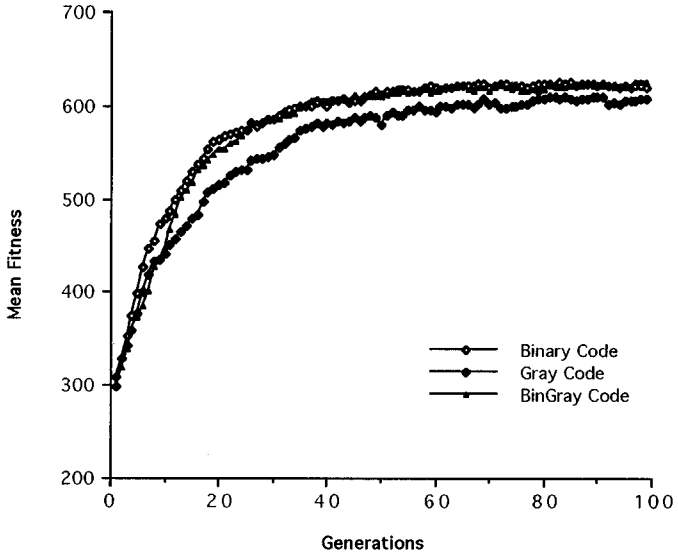


Figure G8.3.3. A comparison of target convergence rate using binary, gray, and bingray codes. Each generation in the simulation has a population of 20 genotype strings, each encoding a phenotypic composite face.

When crossover rate and mutation rate are expressed as single parameters, they may be optimized for any specific application by using a *metalevel* genetic algorithm. In this procedure, metalevel binary strings specify the mutation and crossover rates used by the base level genetic algorithm. In the FacePrints application, a metalevel genetic algorithm evaluated each metastring by determining how well a simulated witness evolved a composite face using the crossover and mutation rates specified by this string. The metalevel strings were evolved over a series of generations to find the optimal parameters for maximizing the efficiency of the search. This resulted in values of 24% for the frequency of crossover and 0.05% for the occurrence of mutations. The simulated witness program was used to evaluate each design modification over hundreds of experimental runs, as well as to evolve the best parameters for the current implementation of the FacePrints program.

G8.3.4 Results

FacePrints has been evaluated experimentally for its ability to generate an accurate facial composite. For these experiments, published composites from the FBI's most-wanted list were used to construct an experimental array of 36 'mug shots'. This 'target array' served as the set of faces from which targets were drawn and from which judges were required to select the culprit when evaluating an experimentally evolved composite. Subjects were required to generate facial composites both when a target face was continuously visible, and after a 30 second exposure to a target, using both a standard recall procedure and the recognition process (FacePrints). In these experiments, the same targets were used by different subjects under the visible, recall, and recognition conditions. The quality of the final composites generated under the three different experimental conditions were evaluated by judges who were required to select the culprit from the 36-face target array, using each of the generated composites. The percentage of correct identifications made by judges on their first attempt served as a measure of the quality of a composite. For judges attempting to identify the culprit using such composites, the mean percentages correct were 70%, 38%, and 60%, for the visible, recall, and recognition composites respectively. Wilcoxon matched-pair signed-rank tests revealed that composites generated under the recognition condition were significantly better than composites generated using the recall procedure ($t(N = 13) = 12$, $p < 0.01$), but significantly worse than composites generated when the same target faces were visible ($t(N = 12) = 14$, $p < 0.05$).

Figure G8.3.4. Target faces of criminals (left) and composites (right) produced by witnesses using the FacePrints procedure.

G8.3.5 Conclusions

Only a small number of published reports have examined the effectiveness of other facial composite systems (Ellis *et al* 1975, Davies *et al* 1978, Ellis *et al* 1986, Laughery and Fowler 1980, Wogalter and Marwitz 1991). These studies indicate that composites generated from recall, using Photofit, Identikit, Mac-A-Mug Pro, or a sketch artist, have serious limits in achieving accurate representations. For example, Ellis *et al* (1975) investigated the usefulness of Photofit with an experimental design similar to that employed in the current study. Photofit composites were generated by subjects immediately following a 10 second exposure to the target face of a culprit. When allowed three attempts, judges selected the culprit from a set of 36 faces on only 25% of the trials. The best performance of a recall-based system was reported by Wogalter and Marwitz (1991). Following an 8 second exposure to a target face, their subjects used Mac-A-Mug Pro to generate a composite face. Judges selected the correct face from an array of six faces with an average success rate of 38%. This performance is comparable to the recall performance in the current study, but is well below the success rate achieved using evolved composites.

FacePrints avoids the problems and limitations of existing face construction systems by adopting a strategy that exploits the well-developed human skill of face recognition. The results indicate that this recognition-based strategy is a significant improvement over processes that depend primarily on recall. The genetic algorithm performs the double function of generating a facial composite and a binary code for that composite. The binary string is similar to a fingerprint, and may serve as a convenient method for searching a database of faces of known criminals to find those that closely resemble the composite code, and to generate a list of potential suspects. Since all facial dimensions, features, and proportions are stored in the code, additional features (e.g. beards, mustaches) and other attributes (e.g. color of complexion) may be added to the database and appended to the binary string with relative ease.

References

Baddeley A D 1979 Applied cognitive and cognitive applied psychology: the case of face recognition *Perspectives on Memory Research* ed L G Nilsson (Hillsdale, NJ: Erlbaum)

Baker J E 1987 Reducing bias and inefficiency in the selection algorithm *Proc. 2nd Int. Conf. on Genetic Algorithms (Cambridge, MA, 1987)* ed J J Grefenstette (Hillsdale, NJ: Erlbaum) pp 14–21

Bethke A D 1981 *Genetic Algorithms as Function Optimizers* Dissertation, Department of Computer and Communication Sciences, University of Michigan

Caldwell C and Johnston V S 1991 Tracking a criminal suspect through 'face space' with a genetic algorithm *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, July 1991)* ed R Belew and L Booker (San Mateo, CA: Morgan Kaufmann) pp 416–21

Caruana R A and Schaffer J D 1988 Representation and hidden bias: gray vs. binary coding for genetic algorithms *Proc. 5th Int. Conf. on Machine Learning (Ann Arbor, MI, 1988)* ed J Laird (San Mateo, CA: Morgan Kaufmann) pp 153–61

Davies G M and Christie D 1982 Face recall: an examination of some factors limiting composite production accuracy *J. Appl. Psychol.* **67** 103–9

Davies G M, Ellis H D and Shepherd J W 1978 Face identification: the influence of delay upon accuracy of Photofit construction *J. Police Sci. Admin.* **6** 35–42

Ellis H D, Davies G M and Shepherd J W 1978 A critical examination of the Photofit system for recalling faces *Ergonomics* **21** 296–307

——1986 Introduction: processes underlying face recognition *The Neuropsychology of Face Perception and Facial Expression* ed R Bruyer (Hillsdale, NJ: Erlbaum) pp 1–38

Ellis H D, Shepherd J W and Davies G M 1975 An investigation of the use of the Photofit technique for recalling faces *Br. J. Psychol.* **66** 29–37

Going M and Read J D 1974 Effects of uniqueness, sex of subject, and sex of photograph on facial recognition *Perceptual Motor Skills* **39** 109–10

Goldberg D E 1989 *Genetic Algorithms in Search, Optimization and Machine Learning* (Boston, MA: Addison-Wesley)

Goldstein A G and Chance J E 1981 Laboratory studies of face recognition *Perceiving and Remembering Faces* ed G Davies, H Ellis and J Shepherd (New York: Academic) pp 81–104

Hall D F 1976 *Obtaining Eyewitness Identification in Criminal Investigations—Applications of Social and Experimental Psychology* Dissertation, Ohio State University

Hines D, Jordan-Brown L and Juzwin K R 1987 Hemispheric visual processing in face recognition *Brain Cognition* **6** 91–100

Holland J H 1975 *Adaptation in Natural and Artificial Systems* (Ann Arbor, MI: University of Michigan Press)

Homa D, Haver B and Schwartz T 1976 Perceptibility of schematic face stimuli: evidence for a perceptual Gestalt *Memory Cognition* **4** 176–85

Johnston V S 1994 *Method and Apparatus for Generating Composites of Human Faces* US Patent 5 375 195

Johnston V S and Franklin M 1993 Is beauty in the eye of the beholder? *Ethol. Sociobiol.* **14** 183–99

Laughery K R and Fowler R H 1980 Sketch artist and Identi-kit procedures for recalling faces *J. Appl. Psychol.* **65** 307–16

Miller L K and Barg M D 1983 Dissociation of feature vs. configural properties in the discrimination of faces *Bull. Psychonom. Soc.* **21** 453–5

Penry J 1974 Photo-Fit *Forens. Photogr.* **3** 4–10

Perkins D A 1975 A definition of caricature and caricature and recognition *Studies Anthropol. Visual Commun.* **2** 1–24

Rakover S S and Cahlon B 1989 To catch a thief with a recognition test: the model and some empirical results *Cognitive Psychol.* **21** 423–68

Ross-Kossak P and Turkewitz G 1986 A micro and macro developmental view of the nature of changes in complex information processing: a consideration of changes in hemispheric advantage during familiarization *The Neuropsychology of Face Perception and Facial Expression* ed R Bruyer (Hillsdale, NJ: Erlbaum) pp 125–45

Solso R L and McCarthy J E 1981 Prototype formation of faces *Br. J. Psychol.* **72** 499–503

Spears W M and De Jong K A 1991 On the virtues of parameterized uniform crossover *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, July 1991)* ed R Belew and L Booker (San Mateo, CA: Morgan Kaufmann) pp 230–6

Suh J Y and Gucht D V 1987 Incorporating heuristic information into genetic search *Proc. 2nd Int. Conf. on Genetic Algorithms (Cambridge, MA, 1987)* ed J J Grefenstette (Hillsdale, NJ: Erlbaum) pp 100–7

Syswerda G 1989 Uniform crossover in genetic algorithms *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, June 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 2–9

Wogalter M S and Marwitz D B 1991 Face composite construction: in-view and from-memory quality and improvement with practice *Ergonomics* **34** 459–68

Yarmey A D and Kent J 1980 Eyewitness identification by elderly and young adults *Law Human Behavior* **4** (special issue) 359–71

Further reading

1. Davies G, Ellis H, and Shepherd J 1981 *Perceiving and Remembering Faces* (New York: Academic)

   An 'introduction' to issues in face recognition and its associated mental processes.

2. Holland J H 1975 *Adaptation in Natural and Artificial Systems* (Ann Arbor, MI: University of Michigan Press)

   The seminal work on the mathematical basis for genetic algorithms.

3. Goldberg D E 1989 *Genetic Algorithms in Search, Optimization and Machine Learning* (Boston, MA: Addison-Wesley)

   An understandable treatment of programming methods using genetic algorithms.

G9.1 Numerical optimization: handling linear constraints

Zbigniew Michalewicz

Abstract

This case study discusses one particular evolutionary method for numerical optimization problems with linear constraints. The method, the Genocop system, is based on maintaining feasible solutions from the convex search space.

G9.1.1 Introduction

Most research on applications of evolutionary computation techniques to numerical optimization problems has been concerned with the following problem:

$$\text{optimize } f(x_1, \ldots, x_n) \in R$$

where $x_k \in \langle l_k, r_k \rangle$ for $1 \leq k \leq n$. Several complex test functions used by various researchers during the last 20 years consider only domains of n variables; this was the case with five test functions F1–F5 proposed by De Jong (1975), as well as with many other test cases proposed since then.

Here we are concerned with the following optimization problem:

$$\text{optimize } f(x_1, \ldots, x_n) \in R$$

where $(x_1, \ldots, x_n) \in \mathcal{D} \subseteq \mathbb{R}^n$ and $\mathcal{D}$ is a *convex* set. The domain $\mathcal{D}$ is defined by ranges of variables ($l_k \leq x_k \leq r_k$ for $k = 1, \ldots, n$) as well as by a set of constraints $\mathcal{C}$. From the convexity of the set $\mathcal{D}$ it follows that for each point in the search space $(x_1, \ldots, x_n) \in \mathcal{D}$ there exists a feasible range $\langle \text{left}(k), \text{right}(k) \rangle$ of a variable x_k ($1 \leq k \leq n$), where other variables x_i ($i = 1, \ldots, k-1, k+1, \ldots, n$) remain fixed. In other words, for a given $(x_1, \ldots, x_k, \ldots, x_n) \in \mathcal{D}$

$$y \in \langle \text{left}(k), \text{right}(k) \rangle \text{iff} (x_1, \ldots, x_{k-1}, y, x_{k+1}, \ldots, x_n) \in \mathcal{D}$$

where all x_i ($i = 1, \ldots, k-1, k+1, \ldots, n$) remain constant. We assume also that the ranges $\langle \text{left}(k), \text{right}(k) \rangle$ can be efficiently computed.

For example, if $\mathcal{D} \subseteq \mathbb{R}^2$ is defined as

$$\begin{aligned} -3 &\leq x_1 \leq 3 \\ 0 &\leq x_2 \leq 8 \\ x_1^2 &\leq x_2 \leq x_1 + 4 \end{aligned}$$

then for a given point $(2, 5) \in \mathcal{D}$

$$\begin{aligned} \text{left}(1) &= 1 \qquad & \text{right}(1) &= \sqrt{5} \\ \text{left}(2) &= 4 \qquad & \text{right}(2) &= 6. \end{aligned}$$

This means that the first component of the vector $(2, 5)$ can vary from 1 to $\sqrt{5}$ (while $x_2 = 5$ remains constant) and the second component of this vector can vary from 4 to 6 (while $x_1 = 2$ remains constant).

Of course, if the set of constraints $\mathcal{C}$ is empty, then the search space $\mathcal{D} = \prod_{k=1}^{n} \langle l_k, r_k \rangle$ is convex; additionally $\text{left}(k) = l_k$, $\text{right}(k) = r_k$ for $k = 1, \ldots, n$.

The Genocop system (for genetic algorithm for numerical optimization of constrained problems, Michalewicz 1996) provides a method of handling linear constraints. Genocop does not use the concept of penalty functions nor does it use a technique of eliminating offspring generated outside the feasible space. The main idea lies in (i) an elimination of the equalities present in the set of constraints, and (ii) careful design of special operators, which guarantee to keep all offspring within the constrained solution space. This can be done very efficiently for linear constraints, which imply convexity of the search space $\mathcal{D}$.

Let us illustrate the mechanism incorporated in the system by the following example. The problem (Hock and Schittkowski 1981) is

$$\text{minimize } f(\boldsymbol{x}) = \sum_{j=1}^{10} x_j \left(c_j + \ln \frac{x_j}{x_1 + \cdots + x_{10}} \right)$$

subject to

$$\begin{aligned} & x_1 + 2x_2 + 2x_3 + x_6 + x_{10} = 2 \\ & x_4 + 2x_5 + x_6 + x_7 = 1 \\ & x_3 + x_7 + x_8 + 2x_9 + x_{10} = 1 \\ & 0.000\,001 \le x_i \le 1.0 \qquad (i = 1, \ldots, 10) \end{aligned}$$

where

$$\begin{aligned} & c_1 = -6.089 \quad c_2 = -17.164 \quad c_3 = -34.054 \quad c_4 = -5.914 \quad c_5 = -24.721 \\ & c_6 = -14.986 \quad c_7 = -24.100 \quad c_8 = -10.708 \quad c_9 = -26.662 \quad c_{10} = -22.179. \end{aligned}$$

It is possible to eliminate three variables, say, x_1, x_3, and x_4 (the newest version of the Genocop system, version 3.0, prompts the user for indices of variables for elimination):

$$\begin{aligned} x_1 &= -2x_2 - x_6 + 2x_7 + 2x_8 + 4x_9 + x_{10} \\ x_3 &= 1 - x_7 - x_8 - 2x_9 - x_{10} \\ x_4 &= 1 - 2x_5 - x_6 - x_7. \end{aligned}$$

Since $0.000\,001 \le x_i \le 1.0$, the transformed problem is to minimize

$$\begin{aligned} & f(x_2, x_5, x_6, x_7, x_8, x_9, x_{10}) \\ & = (-2x_2 - x_6 + 2x_7 + 2x_8 + 4x_9 + x_{10}) \left(c_1 + \ln \frac{-2x_2 - x_6 + 2x_7 + 2x_8 + 4x_9 + x_{10}}{2 - x_2 - x_5 - x_6 + x_7 + 2x_8 + 3x_9 + x_{10}} \right) \\ & \quad + x_2 \left(c_2 + \ln \frac{x_2}{2 - x_2 - x_5 - x_6 + x_7 + 2x_8 + 3x_9 + x_{10}} \right) \\ & \quad + (1 - x_7 - x_8 - 2x_9 - x_{10}) \left(c_3 + \ln \frac{1 - x_7 - x_8 - 2x_9 - x_{10}}{2 - x_2 - x_5 - x_6 + x_7 + 2x_8 + 3x_9 + x_{10}} \right) \\ & \quad + (1 - 2x_5 - x_6 - x_7) \left(c_4 + \ln \frac{1 - 2x_5 - x_6 - x_7}{2 - x_2 - x_5 - x_6 + x_7 + 2x_8 + 3x_9 + x_{10}} \right) \\ & \quad + \sum_{j=5}^{10} x_j \left(c_j + \ln \frac{x_j}{2 - x_2 - x_5 - x_6 + x_7 + 2x_8 + 3x_9 + x_{10}} \right) \end{aligned}$$

subject to

$$\begin{aligned} & 0.000\,001 \le -2x_2 - x_6 + 2x_7 + 2x_8 + 4x_9 + x_{10} \le 1.0 \\ & 0.000\,001 \le x_3 = 1 - x_7 - x_8 - 2x_9 - x_{10} \le 1.0 \\ & 0.000\,001 \le x_4 = 1 - 2x_5 - x_6 - x_7 \le 1.0 \\ & 0.000\,001 \le x_i \le 1.0 \qquad \text{for } i = 2, 5, 6, 7, 8, 9, 10. \end{aligned}$$

After these straightforward transformations, the system creates an initial population of feasible individuals (if, in some predefined number of attempts the system fails to find a feasible point, it will prompt the user for a feasible starting point. In such a case the initial population consists of identical copies of this individual).

There are several operators that alter the composition of individual vectors in the population. We discuss them in turn.

G9.1.2 Uniform mutation

This operator requires a single parent $\boldsymbol{x}$ and produces a single offspring $\boldsymbol{x}'$. The operator selects a random component $k \in (1, \ldots, n)$ of the vector $\boldsymbol{x} = (x_1, \ldots, x_k, \ldots, x_n)$ and produces $\boldsymbol{x}' = (x_1, \ldots, x'_k, \ldots, x_n)$, where x'_k is a random value (uniform probability distribution) from the range $\langle \text{left}(k), \text{right}(k) \rangle$.

The operator plays an important role in the early phases of the evolution process as the solutions are allowed to move freely within the search space. In particular, the operator is essential in the case where the initial population consists of multiple copies of the same (feasible) point. Also, in the later phases of an evolution process the operator allows possible movement away from a local optimum in the search for a better point.

In particular, if the third component (i.e. x_6) of the feasible point (earlier example)

$$(x_2, x_5, x_6, x_7, x_8, x_9, x_{10}) = (0.28, 0.31, 0.11, 0.01, 0.01, 0.21, 0.44)$$

undergoes uniform mutation, then the constraints

$$\begin{aligned} &0.000\,001 \leq x_6 \leq 1.0 \\ &0.000\,001 \leq -2x_2 - x_6 + 2x_7 + 2x_8 + 4x_9 + x_{10} \leq 1.0 \\ &0.000\,001 \leq x_4 = 1 - 2x_5 - x_6 - x_7 \leq 1.0 \end{aligned}$$

imply

$$\begin{aligned} &0.000\,001 \leq x_6 \leq 1.0 \\ &-1.0 - 2x_2 + 2x_7 + 4x_9 + x_{10} \leq x_6 \leq -0.000\,001 - 2x_2 + 2x_7 + 4x_9 + x_{10} \\ &-2x_5 - x_7 \leq x_6 \leq 0.999\,999 - 2x_5 - x_7. \end{aligned}$$

In such a case,

$$\begin{aligned} &0.000\,001 \leq x_6 \leq 1.0 \\ &-0.26 \leq x_6 \leq 0.739\,999 \\ &-0.63 \leq x_6 \leq 0.369\,999 \end{aligned}$$

so

$$\text{left}(3) = 0.000\,001 \qquad \text{and} \qquad \text{right}(3) = 0.369\,999$$

and consequently the third component x_6 of the point $(x_2, x_5, x_6, x_7, x_8, x_9, x_{10})$ acquires a new value (uniform distribution) from the range $\langle 0.000\,001, 0.369\,999 \rangle$.

G9.1.3 Boundary mutation

This operator requires also a single parent $\boldsymbol{x}$ and produces a single offspring $\boldsymbol{x}'$. The operator is a variation of the uniform mutation with x'_k being either $\text{left}(k)$ or $\text{right}(k)$, each with equal probability.

The operator is constructed for optimization problems where the optimal solution lies either on or near the boundary of the feasible search space. Consequently, if the set of constraints $\mathcal{C}$ is empty, and the bounds for variables are quite wide, the operator is a nuisance, but it can prove extremely useful in the presence of constraints.

If the third component x_6 of the point $(x_2, x_5, x_6, x_7, x_8, x_9, x_{10})$ undergoes boundary mutation, it acquires a new value which is (with equal probability) either 0.000 001 or 0.369 999.

G9.1.4 Non-uniform mutation

This is the (unary) operator responsible for the fine-tuning capabilities of the system. It is defined as follows. For a parent $\boldsymbol{x}$, if the element x_k is selected for this mutation, the result is $\boldsymbol{x}' = (x_1, \ldots, x'_k, \ldots, x_n)$, where

$$x'_k = \begin{cases} x_k + \triangle(t, \text{right}(k) - x_k) & \text{if a random binary digit is 0} \\ x_k - \triangle(t, x_k - \text{left}(k)) & \text{if a random binary digit is 1.} \end{cases}$$

The function $\triangle(t, y)$ returns a value in the range $[0, y]$ such that the probability of $\triangle(t, y)$ being close to 0 increases as t increases (t is the generation number). This property causes this operator to search the space uniformly initially (when t is small), and very locally at later stages. We have used the following function:

$$\triangle(t, y) = yr\left(1 - \frac{t}{T}\right)^b$$

where r is a random number from [0..1], T is the maximal generation number, and b is a system parameter determining the degree of nonuniformity. The operator has proven to be extremely useful in many test cases (Michalewicz 1996). See also Section E1.2 for additional discussion on mutation parameters. E1.2

G9.1.5 Arithmetical crossover

This binary operator is defined as a linear combination of two vectors: if $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ are to be crossed, the resulting offspring are $\boldsymbol{x}'_1 = a\boldsymbol{x}_1 + (1 - a)\boldsymbol{x}_2$ and $\boldsymbol{x}'_2 = a\boldsymbol{x}_2 + (1 - a)\boldsymbol{x}_1$. This operator uses a random value $a \in [0..1]$, as it always guarantees closedness ($\boldsymbol{x}'_1, \boldsymbol{x}'_2 \in \mathcal{D}$). Such a crossover has been called a *guaranteed average crossover* (Davis 1989) (when $a = 1/2$); *intermediate crossover* (Bäck *et al* 1991); *linear crossover* (Wright 1991); and *arithmetical crossover* (Michalewicz *et al* 1991).

G9.1.6 Simple crossover

This binary operator is defined as follows: if $\boldsymbol{x}_1 = (x_1, \ldots, x_n)$ and $\boldsymbol{x}_2 = (y_1, \ldots, y_n)$ are crossed after the kth position, the resulting offspring are $\boldsymbol{x}'_1 = (x_1, \ldots, x_k, y_{k+1}, \ldots, y_n)$ and $\boldsymbol{x}'_2 = (y_1, \ldots, y_k, x_{k+1}, \ldots, x_n)$. Such an operator may produce offspring outside the domain $\mathcal{D}$. To avoid this, we use the property of the convex spaces stating that there exists $a \in [0, 1]$ such that

$$\boldsymbol{x}'_1 = (x_1, \ldots, x_k, y_{k+1}a + x_{k+1}(1 - a), \ldots, y_n a + x_n(1 - a))$$

and

$$\boldsymbol{x}'_2 = (y_1, \ldots, y_k, x_{k+1}a + y_{k+1}(1 - a), \ldots, x_n a + y_n(1 - a))$$

are feasible.

The only remaining question to be answered is how to find the largest a. The simplest method would start with $a = 1$ and, if at least one of the offspring does not belong to $\mathcal{D}$, decrease a by some constant $1/q$. After q attempts $a = 0$ and both offspring are in $\mathcal{D}$ since they are identical to their parents. The necessity for such maximal decrement is small in general and decreases rapidly over the life of the population.

G9.1.7 Heuristic crossover

This operator (Wright 1991) is a unique crossover for the following reasons: (i) it uses values of the objective function in determining the direction of the search, (ii) it produces only one offspring, and (iii) it may produce no offspring at all.

The operator generates a single offspring $\boldsymbol{x}_3$ from two parents, $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, according to the following rule:

$$\boldsymbol{x}_3 = r(\boldsymbol{x}_2 - \boldsymbol{x}_1) + \boldsymbol{x}_2$$

where r is a random number between 0 and 1, and the parent $\boldsymbol{x}_2$ is not worse than $\boldsymbol{x}_1$, that is, $f(\boldsymbol{x}_2) \geq f(\boldsymbol{x}_1)$ for maximization problems and $f(\boldsymbol{x}_2) \leq f(\boldsymbol{x}_1)$ for minimization problems.

It is possible for this operator to generate an offspring vector which is not feasible. In such a case another random value r is generated and another offspring created. If after w attempts no new solution meeting the constraints is found, the operator gives up and produces no offspring.

It seems that heuristic crossover contributes towards the precision of the solution found; its major responsibilities are (i) fine local tuning, and (ii) search in the promising direction.

G9.1.8 Test cases

In order to evaluate the method, a set of test problems has been carefully selected to illustrate the performance of the algorithm and to indicate its degree of success. The eight test cases include quadratic, nonlinear, and discontinuous functions with several linear constraints. We used the following parameters for all experiments: population size $\mu = 70$; each operator was applied four times in each generation; and $b = 6$ (coefficient for nonuniform mutation). Genocop was executed ten times for each test case. For most problems, the maximum number of generations T was either 500 or 1000 (harder problems required a larger number of iterations).

Test case no 1. The problem (Floudas and Pardalos 1992) is

$$\text{minimize } f(\boldsymbol{x}, y) = -10.5x_1 - 7.5x_2 - 3.5x_3 - 2.5x_4 - 1.5x_5 - 10y - 0.5\sum_{i=1}^{5} x_i^2$$

subject to

$$\begin{aligned} &6x_1 + 3x_2 + 3x_3 + 2x_4 + x_5 \leq 6.5 \\ &10x_1 + 10x_3 + y \leq 20 \\ &0 \leq x_i \leq 1 \qquad 0 \leq y. \end{aligned}$$

The global solution is $(\boldsymbol{x}^*, y^*) = (0, 1, 0, 1, 1, 20)$, and $f(\boldsymbol{x}^*, y^*) = -213$. Genocop found the optimum in all ten runs.

Test case no 2. The problem (Hock and Schittkowski 1981) is

$$\text{minimize } f(\boldsymbol{x}) = \sum_{j=1}^{10} x_j \left(c_j + \ln \frac{x_j}{x_1 + \cdots + x_{10}} \right)$$

subject to

$$\begin{aligned} &x_1 + 2x_2 + 2x_3 + x_6 + x_{10} = 2 \\ &x_4 + 2x_5 + x_6 + x_7 = 1 \\ &x_3 + x_7 + x_8 + 2x_9 + x_{10} = 1 \\ &x_i \geq 0.000\,001 \qquad (i = 1, \ldots, 10) \end{aligned}$$

where

$$\begin{aligned} &c_1 = -6.089 \quad c_2 = -17.164 \quad c_3 = -34.054 \quad c_4 = -5.914 \quad c_5 = -24.721 \\ &c_6 = -14.986 \quad c_7 = -24.100 \quad c_8 = -10.708 \quad c_9 = -26.662 \quad c_{10} = -22.179. \end{aligned}$$

The best solution reported by Hock and Schittkowski (1981) was

$$\begin{aligned} \boldsymbol{x} = (&0.017\,735\,48, 0.082\,001\,80, 0.882\,564\,6, 0.000\,723\,325\,6, 0.490\,785\,1, \\ &0.000\,433\,546\,9, 0.017\,272\,98, 0.007\,765\,639, 0.019\,849\,29, 0.052\,698\,26) \end{aligned}$$

where $f(\boldsymbol{x}) = -47.707\,579$.

Genocop found points with better value than the one above in all ten runs:

$$\begin{aligned} \boldsymbol{x} = (&0.040\,347\,85, 0.153\,869\,76, 0.774\,970\,89, 0.001\,674\,79, 0.484\,685\,39, \\ &0.000\,689\,65, 0.028\,264\,79, 0.018\,491\,79, 0.038\,495\,63, 0.101\,281\,26) \end{aligned}$$

for which the value of the objective function is equal to $-47.760\,765$.

Test case no 3. The problem (Floudas and Pardalos 1992) is

$$\text{minimize } f(\boldsymbol{x}, \boldsymbol{y}) = 5x_1 + 5x_2 + 5x_3 + 5x_4 - 5\sum_{i=1}^{4} x_i^2 - \sum_{i=1}^{9} y_i$$

subject to

$$\begin{aligned}
&2x_1 + 2x_2 + y_6 + y_7 \leq 10 \qquad 2x_1 + 2x_3 + y_6 + y_8 \leq 10\\
&2x_2 + 2x_3 + y_7 + y_8 \leq 10 \qquad -8x_1 + y_6 \leq 0\\
&-8x_2 + y_7 \leq 0 \qquad -8x_3 + y_8 \leq 0\\
&-2x_4 - y_1 + y_6 \leq 0 \qquad -2y_2 - y_3 + y_7 \leq 0\\
&-2y_4 - y_5 + y_8 \leq 0 \qquad 0 \leq x_i \leq 1 \quad (i = 1, 2, 3, 4)\\
&0 \leq y_i \leq 1 \quad (i = 1, 2, 3, 4, 5, 9) \qquad 0 \leq y_i \quad (i = 6, 7, 8).
\end{aligned}$$

The global solution is $(\boldsymbol{x}^*, \boldsymbol{y}^*) = (1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 3, 3, 1)$, and $f(\boldsymbol{x}^*, \boldsymbol{y}^*) = -15$. Genocop found the optimum in all ten runs.

Test case no 4. The problem (Floudas and Pardalos 1987) is

$$\text{maximize } f(\boldsymbol{x}) = \frac{3x_1 + x_2 - 2x_3 + 0.8}{2x_1 - x_2 + x_3} + \frac{4x_1 - 2x_2 + x_3}{7x_1 + 3x_2 - x_3}$$

subject to

$$\begin{aligned}
&x_1 + x_2 - x_3 \leq 1 \qquad -x_1 + x_2 - x_3 \leq -1\\
&12x_1 + 5x_2 + 12x_3 \leq 34.8 \qquad 12x_1 + 12x_2 + 7x_3 \leq 29.1\\
&-6x_1 + x_2 + x_3 \leq -4.1 \qquad 0 \leq x_i \quad (i = 1, 2, 3).
\end{aligned}$$

The global solution is $\boldsymbol{x}^* = (1, 0, 0)$, and $f(\boldsymbol{x}^*) = 2.471\,428$. Genocop found the optimum in all ten runs.

Test case no 5. The problem (Floudas and Pardalos 1992) is

$$\text{minimize } f(\boldsymbol{x}) = x_1^{0.6} + x_2^{0.6} - 6x_1 - 4x_3 + 3x_4$$

subject to

$$\begin{aligned}
&-3x_1 + x_2 - 3x_3 = 0 \qquad x_1 + 2x_3 \leq 4\\
&x_2 + 2x_4 \leq 4 \qquad x_1 \leq 3 \qquad x_4 \leq 1\\
&0 \leq x_i \quad (i = 1, 2, 3, 4).
\end{aligned}$$

The global solution is $\boldsymbol{x}^* = (\frac{4}{3}, 4, 0, 0)$, and $f(\boldsymbol{x}^*) = -4.5142$. Genocop found this point in all ten runs.

Test case no 6. The problem (Colville 1968) is

$$\begin{aligned}
\text{minimize } f(\boldsymbol{x}) = {} & 100(x_2 - x_1^2)^2 + (1 - x_1)^2 + 90(x_4 - x_3^2)^2 + (1 - x_3)^2\\
& + 10.1((x_2 - 1)^2 + (x_4 - 1)^2) + 19.8(x_2 - 1)(x_4 - 1)
\end{aligned}$$

subject to

$$-10.0 \leq x_i \leq 10.0 \quad i = 1, 2, 3, 4.$$

The global solution is $\boldsymbol{x}^* = (1, 1, 1, 1)$, and $f(\boldsymbol{x}^*) = 0$.

Genocop approached the optimum quite closely in all ten runs; a typical optimum point found was

$$(0.999\,999\,64, 0.999\,999\,28, 1.000\,000\,36, 1.000\,000\,72)$$

for which the value of the objective function is equal to $0.000\,000\,000\,000\,466\,560$. However, to get this precision, the number of generations was set at 100 000.

Test case no 7. The problem (Floudas and Pardalos 1992) is

$$\text{minimize } f(x, \boldsymbol{y}) = 6.5x - 0.5x^2 - y_1 - 2y_2 - 3y_3 - 2y_4 - y_5$$

subject to

$$\begin{aligned}
&x + 2y_1 + 8y_2 + y_3 + 3y_4 + 5y_5 \leq 16\\
&-8x - 4y_1 - 2y_2 + 2y_3 + 4y_4 - y_5 \leq -1\\
&2x + 0.5y_1 + 0.2y_2 - 3y_3 - y_4 - 4y_5 \leq 24\\
&0.2x + 2y_1 + 0.1y_2 - 4y_3 + 2y_4 + 2y_5 \leq 12\\
&-0.1x - 0.5y_1 + 2y_2 + 5y_3 - 5y_4 + 3y_5 \leq 3\\
&y_3 \leq 1 \qquad y_4 \leq 1 \qquad y_5 \leq 2 \qquad x \geq 0\\
&y_i \geq 0 \qquad \text{for } 1 \leq i \leq 5.
\end{aligned}$$

The global solution is $(x, \boldsymbol{y}^*) = (0, 6, 0, 1, 1, 0)$, and $f(x, \boldsymbol{y}^*) = -11$. Genocop found the optimum in all ten runs.

Test case no 8. The problem was constructed from three separate problems (Hock and Schittkowski 1981) in the following way:

$$\text{minimize } f(\boldsymbol{x}) = \begin{cases} f_1 = x_2 + 10^{-5}(x_2 - x_1)^2 - 1.0 & \text{if } 0 \leq x_1 < 2\\ f_2 = \frac{1}{27\,(3)^{1/2}}((x_1 - 3)^2 - 9)x_2^3 & \text{if } 2 \leq x_1 < 4\\ f_3 = \frac{1}{3}(x_1 - 2)^3 + x_2 - \frac{11}{3} & \text{if } 4 \leq x_1 \leq 6 \end{cases}$$

subject to

$$\begin{aligned}
&x_1/3^{1/2} - x_2 \geq 0\\
&-x_1 - 3^{1/2}x_2 + 6 \geq 0\\
&0 \leq x_1 \leq 6 \qquad x_2 \geq 0.
\end{aligned}$$

The function f has three global solutions:

$$\boldsymbol{x}_1^* = (0, 0) \qquad \boldsymbol{x}_2^* = (3, 3^{1/2}) \qquad \text{and} \qquad \boldsymbol{x}_3^* = (4, 0).$$

In all cases $f(\boldsymbol{x}_i^*) = -1$ $(i = 1, 2, 3)$.

We performed three separate experiments. In experiment k $(k = 1, 2, 3)$ all functions f_i except f_k were increased by 0.5. As a result, the global solution for the first experiment was $\boldsymbol{x}_1^* = (0, 0)$, the global solution for the second experiment was $\boldsymbol{x}_2^* = (3, 3^{1/2})$, and the global solution for the third experiment was $\boldsymbol{x}_3^* = (4, 0)$. Genocop found global optima in all runs in all three cases.

The Genocop system is available from anonymous ftp unccsun.uncc.edu, directory coe/evol, file genocop3.0.tar.Z.

References

Bäck T, Hoffmeister F and Schwefel H-P 1991 A survey of evolution strategies *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, July 1991)* ed R K Belew and L Booker (San Mateo, CA: Morgan Kaufmann) pp 2–9

Colville A R 1968 *A Comparative Study on Nonlinear Programming Codes* IBM Scientific Center Report 320-2949

Davis L 1989 Adapting operator probabilities in genetic algorithms *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, June 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 61–9

Floudas C A and Pardalos P M 1987 *A Collection of Test Problems for Constrained Global Optimization Algorithms* (*Lecture Notes in Computer Science 455*) (Berlin: Springer)

——1992 *Recent Advances in Global Optimization* (*Princeton Series in Computer Science*) (Princeton, NJ: Princeton University Press)

Hock W and Schittkowski K 1981 *Test Examples for Nonlinear Programming Codes* (*Lecture Notes in Economics and Mathematical Systems 187*) (Berlin: Springer)

Michalewicz Z 1996 *Genetic Algorithms + Data Structures = Evolution Programs* 3rd edn (New York: Springer)

Michalewicz Z, Vignaux G A and Hobbs M 1991 A non-standard genetic algorithm for the nonlinear transportation problem *ORSA J. Comput.* **3** 307–16

Wright A H 1991 Genetic algorithms for real parameter optimization *Foundations of Genetic Algorithms, 1st Workshop on the Foundations of Genetic Algorithms and Classifier Systems* ed G Rawlins (San Mateo, CA: Morgan Kaufmann) pp 205–18

G9.2 A genetic algorithm approach to the maximum independent set problem

Thomas Bäck

Abstract

The results obtained from the application of a genetic algorithm to the NP-complete maximum independent set problem are reported in this work. In contrast to many other genetic algorithm-based approaches that use domain-specific knowledge, the approach presented here relies on a graded penalty term component of the objective function to penalize infeasible solutions. The method is applied to several large problem instances of the maximum independent set problem, and the results clearly indicate that genetic algorithms can be successfully used as heuristics for finding good approximate solutions for this highly constrained optimization problem.

G9.2.1 Introduction

An approach that uses genetic algorithms as a generalized heuristic for finding approximate solutions of NP-hard combinatorial optimization problems is presented for the maximum independent set problem. This work (Bäck and Khuri 1994) is part of a series of investigations regarding the usefulness of the genetic algorithm as a heuristic for combinatorial optimization problems (see also Khuri *et al* 1994a, 1994b, Khuri and Bäck 1994 and Bäck *et al* 1996).

G9.2.2 The maximum independent set problem

The maximum independent set problem consists of finding the largest subset of vertices of a graph such that none of these vertices are connected by an edge (i.e. the vertices are independent of each other). Let $G = (V, E)$ denote a graph where V is the set of nodes and $E \subseteq V \times V$ is the set of edges. The problem is to determine a set $V' \subseteq V$ such that $\forall i, j \in V'$ the edge $\langle i, j \rangle \notin E$ and $|V'|$ is maximum. The problem is known to be NP-complete (see Garey and Johnson 1979, pp 53–6).

With the maximum independent set problem, one also obtains a solution to two other important graph problems: the minimum vertex cover problem (which has important applications in matching problems) and the maximum clique problem. The minimum vertex cover problem consists of finding the smallest subset $V' \subseteq V$ such that $\forall \langle i, j \rangle \in E : i \in V' \vee j \in V'$ (the smallest set of vertices that covers all edges), while in case of the maximum clique problem the goal is to find the largest subset $V' \subseteq V$ such that $\forall i, j \in V' : \langle i, j \rangle \in E$. The following lemma clarifies the close relationship between these problems (Garey and Johnson 1979):

Lemma 1. For any graph $G = (V, E)$ and $V' \subseteq V$, the following statements are equivalent:

- V' is the maximum independent set in G.
- $V - V'$ is the minimum vertex cover of G.
- $V - V'$ is the maximum clique in $G^{\mathrm{C}} = (V, E^{\mathrm{C}})$, where $E^{\mathrm{C}} = \{ \langle i, j \rangle \mid i, j \in V \wedge \langle i, j \rangle \notin E \}$.

Consequently, one can obtain a solution of the minimum vertex cover problem by taking the complement of the solution to the maximum independent set problem. A solution to the maximum clique problem is obtained by applying the maximum independent set heuristic to $G^{\mathrm{C}} = (V, E^{\mathrm{C}})$.

The lemma clarifies that the maximum independent set problem and minimum vertex cover problem are dual problems, and it is simple to transfer results obtained for one problem to the other one (Khuri and Bäck (1994) presented a similar approach for the minimum vertex cover problem).

Rather than using pairs $\langle i, j\rangle \in V \times V$ of node numbers from $V = \{1, \ldots, \ell\}$, a graph is represented in the following by its $\ell \times \ell$ adjacency matrix (e_{ij}), where

$$e_{ij} = \begin{cases} 1 & \text{if } \langle i, j\rangle \in E \\ 0 & \text{otherwise.} \end{cases} \tag{G9.2.1}$$

Using terminology from Stinson (1987) for combinatorial optimization problems, the maximum independent set problem is formulated as follows:

Problem instance: A graph $G = (V, E)$ with vertices $V = \{1, \ldots, \ell\}$ and edges $E \subseteq V \times V$, represented by the adjacency matrix (e_{ij}).

Feasible solution: A set V' of nodes such that $\forall i, j \in V' : e_{ij} = 0$.

Objective function: The size $|V'|$ of the independent set V'.

Optimal solution: An independent set V' that maximizes $|V'|$.

G9.2.3 Design process

In order to encode the problem for a genetic algorithm, we choose the following representation of a candidate solution as a *binary string* $(b_1, b_2, \ldots, b_\ell) \in \{0, 1\}^\ell$: $b_i = 1 \Leftrightarrow i \in V'$. This way, the ith bit indicates the presence ($b_i = 1$) or absence ($b_i = 0$) of vertex i in the candidate solution. Note that a bitstring may (and often will) represent an infeasible solution. Instead of trying to prevent this, we allow infeasible strings to join the population and use a *penalty function* approach to guide the search towards the feasible region (Richardson *et al* 1989). The penalty term in the objective function has to be graded in the sense that the farther away from feasibility the string is, the larger its penalty term. The exact nature of the penalty function, however, is not critically important so long as it fulfills the property of being graded (see Smith and Tate 1993). C1.2 C5.2

Taking this design rule for a penalty function into consideration, we developed the following objective function to be maximized by the genetic algorithm:

$$f(\boldsymbol{b}) = \sum_{i=1}^{\ell} \left(b_i - \ell b_i \sum_{j=i}^{\ell} b_j e_{ij} \right). \tag{G9.2.2}$$

This function penalizes infeasible strings $\boldsymbol{b}$ by a penalty of ℓ for every node j in the candidate solution V' represented by $\boldsymbol{b}$ that is connected to a node $i \in V'$. For feasible strings $\boldsymbol{b}$, $f(\boldsymbol{b}) \geq 0$ and the function value is just given by the number of nodes in the independent set represented by $\boldsymbol{b}$.

Based on this representation and the objective function given in (G9.2.2), a canonical *genetic algorithm* is directly applicable to the problem. For the experiments reported here, we used: B1.2

- a *population size* of $\mu = 50$ individuals; E1.1
- a *mutation rate* $p_{\mathrm{m}} = 1/\ell$, motivated by theoretical results (see Bäck 1992 or Mühlenbein 1992); E1.2
- *two-point crossover* with a *crossover rate* $p_c = 0.6$; and C3.3.1, E1.3
- *proportional selection* with linear dynamic scaling (and a scaling window of five generations). C2.2

G9.2.4 Development and implementation

The experimental runs are performed with the genetic algorithm software package GENEsYs 2.0 (Bäck 1996), which is based on Grefenstette's GENESIS (see Davis 1991, pp 374–7)) but offers more flexibility of the genetic operators and the data monitoring features.

In order to obtain large test problems for an application of the genetic algorithm to the maximum independent set problem, we make use of the scalable graph shown in figure G9.2.1, which can be constructed for an even number of nodes ℓ ($\ell \geq 6$). If ℓ is a multiple of 4, two equivalent global maxima of function value $|V'| = \ell/2$ are obtained by partitioning the set of vertices into those of even and those of odd node numbers. Otherwise, the unique global maximum is given by $V' = \{1, 3, \ldots, \ell/2, \ell/2 + 1, \ell/2 + 3, \ldots, \ell\}$, with objective function value $\ell/2 + 1$, and a local maximum is obtained from $V - V'$ with objective function value $\ell/2 - 1$. For $\ell = 10$, the corresponding bitstrings are $\boldsymbol{b}' = (1010110101)$ and its inverted form (0101001010) (see figure G9.2.1). Notice that these optima are separated from each other by the maximum Hamming distance, which is possible, that is, by a distance of ℓ.

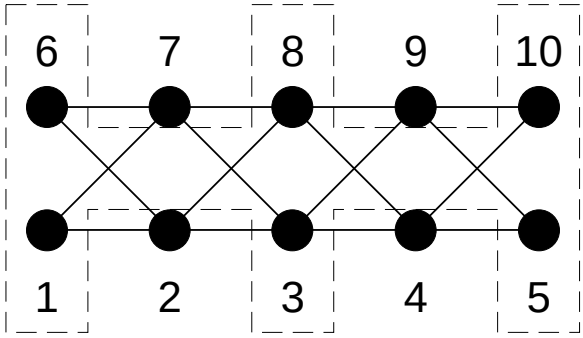


Figure G9.2.1. Example graph 'misp10' with $\ell = 10$ nodes. The independent set $V' = \{1, 3, 5, 6, 8, 10\}$, represented by the bitstring (1010110101), is indicated by the dashed lines. Notice that the independent set $\{2, 4, 7, 9\}$, represented by the bitstring (0101001010), is a local maximum. Reprinted by permission of IEEE from Bäck and Khuri (1994, volume II, p 532, figure 1, copyright 1994 IEEE).

In addition to this graph, which has a highly regular structure, we use randomly constructed graphs which are created according to the following algorithm with input $k \in \{1, \ldots, \ell\}$ (number of nodes in V') and $d \in [0, 1]$ (edge density of the graph):

Input: $k \in \{1, \ldots, \ell\}$, $d \in [0, 1]$
Output: $E = (e_{ij})$
1 randomly select $V' = \{i_1, \ldots, i_k\} \subseteq V = \{1, \ldots, \ell\}$;
2 **for** $i \leftarrow 1$ **to** ℓ **do**
3 **for** $j \leftarrow i + 1$ **to** ℓ **do**
4 **if** $((u \sim U([0, 1]) < d)$ **and** $((i \notin V')$ **or** $(j \notin V')))$
then $e_{ij} = 1$;
else $e_{ij} = 0$;
5 **return** (e_{ij});

The algorithm randomly preselects k nodes $i_1, \ldots, i_k$ (line 1) that are guaranteed to form an independent set (the graph may, however, contain different larger independent sets by chance, especially when the edge density is low). Edges are placed at random (line 4), according to the density parameter d, such that it is guaranteed that a member of V' is never connected to another member of V' (line 4). Note that, according to the construction method, only loop-free graphs are generated.

For the experimental test, regular graphs of size $\ell = 102$ and $\ell = 202$ (with a maximum independent set of size 52 and 102, respectively) and random graphs with $\ell = 100$, $k = 45$, and $\ell = 200$, $k = 90$, respectively, and $d \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$ are used. By choosing $\ell = 102$ and $\ell = 202$ for the regular graph, we work with graphs where an almost optimal local optimum exists in addition to the global optimum. For each of these problems, a total of $N = 100$ runs of the genetic algorithm are performed. These runs are evaluated according to the number of runs that yield solutions of identical quality.

G9.2.5 Results

The results are summarized in table G9.2.1 (for the regular/random graphs with 102/100 vertices) and table G9.2.2 (for the regular/random graphs with 202/200 vertices) for the best results that were encountered during the 100 runs for each test problem. For each experiment, the average final best function value $\bar{f}$ over

Table G9.2.1. Experimental results for the regular graph 'misp102' with $\ell = 102$ vertices and five random graphs with edge density $d = 0.1$ ('misp100-01'), $d = 0.2$ ('misp100-02'), $d = 0.3$ ('misp100-03'), $d = 0.4$ ('misp100-04') and $d = 0.5$ ('misp100-05'). An independent set size $k = 45$ was chosen for the random graphs, but for the graph with $d = 0.1$ the genetic algorithm identified a larger independent set. Reprinted by permission of IEEE from Bäck and Khuri (1994, volume II, p 533, table 1, copyright 1994 IEEE).

misp102		misp100-01		misp100-02		misp100-03		misp100-04		misp100-05	
$f_{2\times10^4}(\mathbf{x})$	N	$f_{2\times10^4}(\mathbf{x})$	N	$f_{2\times10^4}(\mathbf{x})$	N	$f_{2\times10^4}(\mathbf{x})$	N	$f_{2\times10^4}(\mathbf{x})$	N	$f_{2\times10^4}(\mathbf{x})$	N
52	—	47	1	45	34	45	77	45	96	45	99
50	1	46	1	43	3	44	1	33	1	17	1
48	14	45	3	41	2	41	6	32	1		
46	32	44	6	40	3	37	3	25	1		
44	40	43	4	39	10	36	3	10	1		
42	10	42	4	38	1	34	1				
40	3	41	9	37	8	33	1				
		40	4	36	1	32	1				
		39	6	35	1	30	1				
		38	12	34	6	29	1				
		37	5	33	6	26	1				
		< 37	45	< 33	25	< 26	4				
$\bar{f} = 44.94$		$\bar{f} = 37.39$		$\bar{f} = 37.25$		$\bar{f} = 42.38$		$\bar{f} = 44.20$		$\bar{f} = 44.72$	

all 100 runs is indicated at the bottom of the table. The total number of function evaluations performed for each single run is indicated as an index t in the notation $f_t(\boldsymbol{x})$; for $\ell = 100$ we use a value of $t = 2 \times 10^4$, while this is doubled for $\ell = 200$. Consequently, only a small percentage of the search space (about 1.6×10^{-24}% for $\ell = 100$ and 2.5×10^{-54}% for $\ell = 200$) is tested by the genetic algorithm.

For the regular graphs 'misp102' and 'misp202', none of the runs of the genetic algorithm identified the globally optimal solutions of quality 52 and 102, respectively, but for all runs a solution quality between 40 and 50 or between 82 and 96, respectively, was obtained, i.e. solutions with a quality close to the optimal one were found. Finding the global optimum in the case of these regular graphs becomes an extremely difficult problem due to large Hamming distances between local optima of similar quality (e.g. consider $f(101001010101001010) = 8$, $f(101010101101010101) = 10$, and the Hamming distance between both strings is 10).

Table G9.2.2. Experimental results for the regular graph 'misp202' with $\ell = 202$ vertices and five random graphs with edge density $d = 0.1$ ('misp200-01'), $d = 0.2$ ('misp200-02'), $d = 0.3$ ('misp200-03'), $d = 0.4$ ('misp200-04') and $d = 0.5$ ('misp200-05'). An independent set size $k = 90$ was chosen for the random graphs. Reprinted by permission of IEEE from Bäck and Khuri (1994, volume II, p 534, table 2, copyright 1994 IEEE).

misp202		misp200-01		misp200-02		misp200-03		misp200-04		misp200-05	
$f_{4\times10^4}(\mathbf{x})$	N	$f_{4\times10^4}(\mathbf{x})$	N	$f_{4\times10^4}(\mathbf{x})$	N	$f_{4\times10^4}(\mathbf{x})$	N	$f_{4\times10^4}(\mathbf{x})$	N	$f_{4\times10^4}(\mathbf{x})$	N
102	—	90	4	90	54	90	93	90	100	90	100
96	3	88	1	89	1	72	1				
94	3	85	2	80	4	70	2				
92	11	84	3	79	6	65	1				
90	33	82	2	78	3	62	2				
88	30	81	2	77	5	51	1				
86	12	80	4	75	4						
84	5	79	1	74	2						
82	3	78	5	73	3						
		77	5	71	1						
		76	1	70	1						
		< 76	70	< 70	16						
$\bar{f} = 88.90$		$\bar{f} = 68.75$		$\bar{f} = 81.05$		$\bar{f} = 88.22$		$\bar{f} = 90.00$		$\bar{f} = 90.00$	

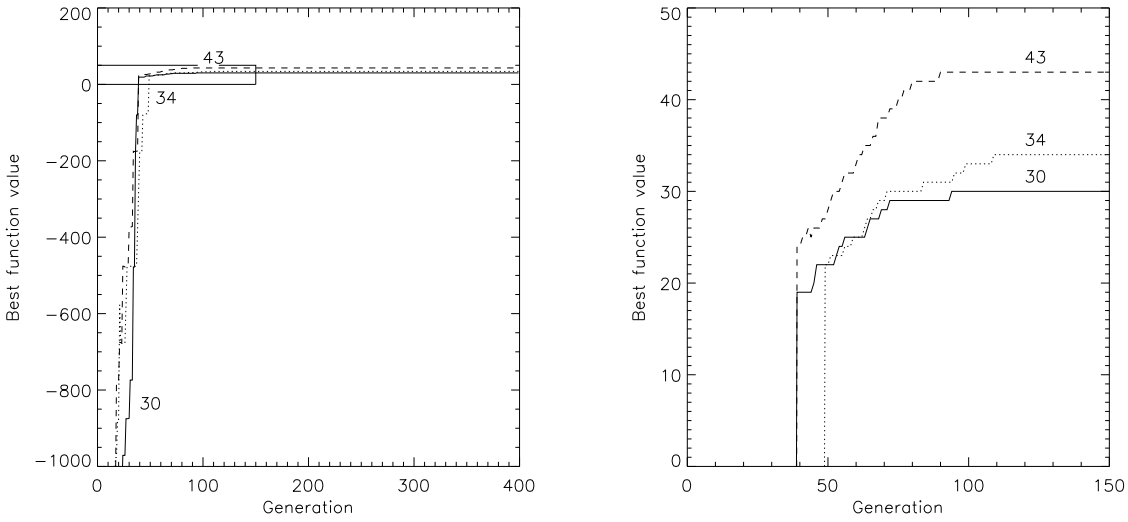


Figure G9.2.2. Some representative courses of evolution for the maximum independent set problem (using a random graph of size $\ell = 100$ with an edge density $d = 0.1$). The left plot shows the complete run, while the right plot shows a magnification of the marked region. Reprinted by permission of IEEE from Bäck and Khuri (1994, volume II, p 534, figure 2 and p 535, figure 3, copyright 1994 IEEE).

A comparison of the results for the random graphs reveals that the edge density is the major factor which determines the complexity of the maximum independent set problem.

The smaller (larger) the edge density, the fewer (more) runs succeed in finding a solution of quality $k = 45$ or $k = 90$, respectively, or better (which is possible in case of small edge density, e.g. for $d = 0.1$). For small edge density, the number of local optima grows due to the possibility of exchanges of groups of vertices and the existence of isolated vertices. As the edge density increases to a value of 0.5, the frequency of runs that identify the solution with 45 or 90 vertices, respectively, grows steadily. For the smaller graphs, the genetic algorithm always found the best solution for an edge density above $d = 0.5$, while this property already holds for the larger graphs for $d = 0.4$.

Notice that, according to the construction mechanism, the edge density of the regular graph is

$$d = \frac{4(\ell - 2)}{\ell(\ell - 1)} \approx 4/\ell \tag{G9.2.3}$$

(the regular graph has $2\ell - 4$ edges, and the maximum number of edges is $\ell(\ell - 1)/2$ if no loops are permitted).

From the experience with random graphs, it is clear that this small value provides further evidence for the complexity of the regular graph problems.

All runs of the genetic algorithm were characterized by the following properties, independently of the problem instance to which the algorithm was applied. The initial phase of the search was used for finding feasible solutions from a completely infeasible initial population. The genetic algorithm quickly succeeded in leaving the infeasible region in each of the runs reported here, thus demonstrating the appropriateness of our graded penalty function approach. After at most 200 or 400 generations, respectively, (1×10^4 or 2×10^4 function evaluations) each run had settled in a local optimum and did not show further improvement. The quality of the optima found, however, clarifies the genetic algorithms' reliability for identifying good approximate solutions for the maximum independent set problems studied here.

To illustrate the typical behavior of genetic algorithm runs, figure G9.2.2 shows the course of evolution for three different runs on the 'misp100-01' problem. The best objective function value that occurred in the population is plotted over the generation number for each of the three runs. Each run is labeled by its final solution quality, and the ordinate axis is restricted to a smaller range of values than really observed (initially best values are found around -3.5×10^3). Note that only about 50 generations are required to enter the feasible region (which corresponds with non-negative function values). Further progress is observed until approximately generation 100, and afterwards the search stagnates in local optima.

The right part of figure G9.2.2 shows a magnification of the marked region from the left plot. This closer look reveals that between generations 50 and 100 a steady period of further improvement of feasible solutions takes place. During this stage of the search, the algorithm fine-tunes solutions towards one of the local optima of the search space.

G9.2.6 Conclusions

We have shown in this work that genetic algorithms can be used in a fairly straightforward way to find good approximate solutions to the NP-hard maximum independent set problem. The robustness of our approach based on a graded penalty function for infeasible strings is demonstrated by the fact that no changes to the genetic algorithm are required. Thus, rather than having to construct tailored heuristics to handle the problem under consideration, we suggest the use of genetic algorithms where the only change to perform is the formulation of a new objective function.

References

Bäck T 1992 The interaction of mutation rate, selection, and self-adaptation within a genetic algorithm *Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature (Brussels, 1992)* ed R Männer and B Manderick (Amsterdam: Elsevier) pp 85–94

——1996 *Evolutionary Algorithms in Theory and Practice* (New York: Oxford University Press)

Bäck T and Khuri S 1994 An evolutionary heuristic for the maximum independent set problem *Proc. 1st. IEEE Int. Conf. on Evolutionary Computation (Orlando, FL, June 1994)* ed Z Michalewicz *et al* (Piscataway, NJ: IEEE Press) pp 531–35

Bäck T, Schütz M and Khuri S 1996 A comparative study of a penalty function, a repair heuristic, and stochastic operators with the set-covering problem *Artificial Evolution* (*Lecture Notes in Computer Science 1063*) ed M Alliot *et al* (Berlin: Springer) pp 320–32

Davis L 1991 *Handbook of Genetic Algorithms* (New York: Van Nostrand Reinhold)

Garey M R and Johnson D S 1979 *Computers and Intractability — A Guide to the Theory of NP-Completeness* (San Francisco, CA: Freemann)

Khuri S and Bäck T 1994 An evolutionary heuristic for the minimum vertex cover problem *Genetic Algorithms within the Framework of Evolutionary Computation (Report MPI-I-94-241)* ed J Hopf (Saarbrücken: Max-Planck-Institut für Informatik) pp 86–90

Khuri S, Bäck T and Heitkötter J 1994a The zero/one multiple knapsack problem and genetic algorithms *Proc. 1994 ACM Symp. on Applied Computing* ed E Deaton *et al* (New York: ACM Press) pp 188–93

——1994b An evolutionary approach to combinatorial optimization problems *Proc. 22nd Annual ACM Computer Science Conf.* ed D Cizmar (New York: ACM Press) pp 66–73

Mühlenbein H 1992 How genetic algorithms really work: I mutation and hillclimbing *Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature (Brussels, 1992)* ed R Männer and B Manderick (Amsterdam: Elsevier) pp 15–25

Richardson J T, Palmer M R, Liepins G and Hilliard M 1989 Some guidelines for genetic algorithms with penalty functions *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, June 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 191–7

Smith A E and Tate D M 1993 Genetic optimization using a penalty function *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 499–505

Stinson D R 1987 *An Introduction to the Design and Analysis of Algorithms* (Winnipeg: Charles Babbage Research Center)

G9.3 An ecosystem model for integrated production planning

Philip Husbands, Malcolm McIlhagga and Robert Ives

Abstract

This case study outlines a coevolutionary distributed genetic algorithm for tackling an integrated manufacturing planning and scheduling problem. In this multispecies ecosystem model, the genotype of each species represents a feasible manufacturing (process) plan for a particular component to be manufactured in the machine shop. Separate populations evolve under the pressure of selection to find near-optimal process plans for each of the components. However, their fitness functions take into account the use of shared resources in their common world (a model of the machine shop). This means that, without the need for an explicit scheduling stage, a low-cost schedule will emerge at the same time as the plans are being optimized. Results are presented of the use of this model on a set of industrial problems. It is shown to significantly outperform simulated annealing and a dispatching rule algorithm over a wide range of optimization criteria.

G9.3.1 Project overview

Research on job shop scheduling (JSS), as the most general of the classical *scheduling problems*, has generated a great deal of literature (Muth and Thomson 1963, Balas 1969, Garey *et al* 1976, Graves 1981, Ow and Smith 1988, Carlier and Pinson 1989). All of this work has used a particular definition of the scheduling problem or very close variants of it. This article describes a case study where a multispecies coevolutionary *genetic algorithm* is used to tackle a less restricted highly generalized version of JSS. It is shown how the technique provides an integrated production planning system, treating process planning, and scheduling as inextricably interwoven parts of the same problem.

F1.5

B1.2

The traditional view of JSS is shown in figure G9.3.1. A number of *fixed* manufacturing plans, one for each component to be manufactured, are interleaved by a scheduler so as to minimize some criteria such as the total length of the schedule. More formally, we are given a set $\mathcal{J}$ of n jobs, a set $\mathcal{M}$ of m machines, and a set $\mathcal{O}$ of K operations. For each operation $p \in \mathcal{O}$ there is one job $j_p \in \mathcal{J}$ to which it belongs, and one machine $m_p \in \mathcal{M}$ on which it must be processed for a time $t_p \in \mathbf{N}$. There is also a binary temporal ordering relation $\rightarrow$ on $\mathcal{O}$ that decomposes the set into partial ordering networks corresponding to the jobs. That is, if $x \rightarrow y$, then $j_x = j_y$ and there is no z, distinct from x and y, such that $x \rightarrow z$ or $z \rightarrow y$. Using the minimize makespan objective function, i.e. minimizing the elapsed time needed to finish processing all jobs, the problem is to find a start time s_p for each operation $p \in \mathcal{O}$ such that

$$\max_{p \in \mathcal{O}}(s_p + t_p) \tag{G9.3.1}$$

is minimized subject to

$$t_p \geq 0 \qquad \forall p \in \mathcal{O} \tag{G9.3.2}$$

$$s_x - s_y \geq t_y \qquad \text{if } y \rightarrow x, \quad x, y \in \mathcal{O} \tag{G9.3.3}$$

$$(s_i - s_j \geq t_j) \bigvee (s_j - s_i \geq t_i) \qquad \text{if } m_i = m_j, \quad i, j \in \mathcal{O}. \tag{G9.3.4}$$

However, a problem that would often be more useful to solve is that illustrated in figure G9.3.2. Here the intention is to optimize the individual manufacturing plans *in parallel*, taking into account the numerous interactions between them resulting from the shared use of resources. This is the optimization task that henceforth will be termed the integrated planning and scheduling problem and is the focus of this case study. An ecosystems model has been developed to tackle various practical instances of this problem, one of which is presented here.

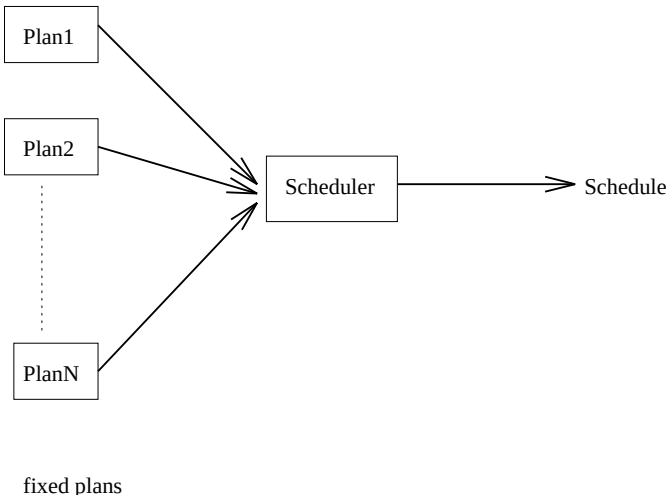


Figure G9.3.1. The traditional approach to job shop sheduling.

The idea behind the ecosystems model is as follows. The genotype of each species represents a feasible manufacturing (process) plan for a particular component to be manufactured in the machine shop. Separate populations evolve under the pressure of selection to find near-optimal process plans for each of the components. However, their fitness functions take into account the use of shared resources in their common world (a model of the machine shop). This means that, without the need for an explicit scheduling stage, a low-cost schedule will emerge at the same time as the plans are being optimized. The system is illustrated in figure G9.3.3. The role of the arbitrators, which coevolve along with the other species, is to resolve resource conflicts between manufacturing plans for different components.

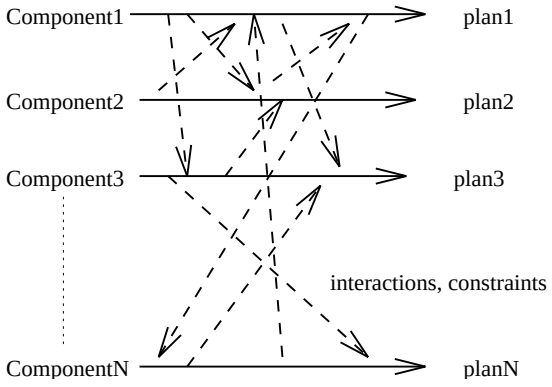


Figure G9.3.2. Parallel plan optimization leading to emergent scheduling.

This project is one of the strands of ongoing research in the Evolutionary and Adaptive Systems Group, School of Cognitive and Computing Sciences, University of Sussex. It has been carried out in collaboration with Edinburgh University, Logica and Rolls Royce.

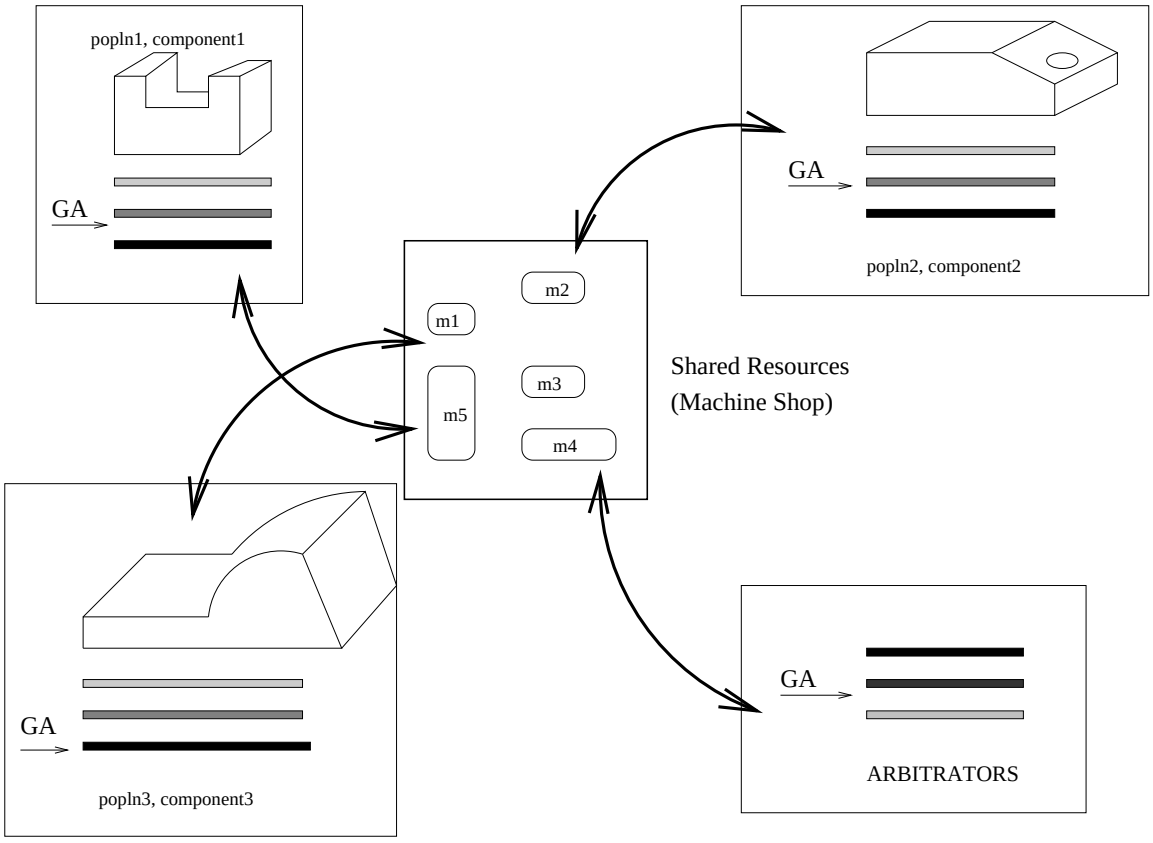


Figure G9.3.3. The ecosystem model.

G9.3.1.1 Description of the problem

The integrated planning and scheduling problems considered in this case study are typical industrial problems. They are generated from data collected from David Brown Vehicle Transmissions Ltd. They model the manufacture of medium-complexity prismatic parts, by metal removal processes. They are based on the work of Palmer (1994).

The statistics shown in section G9.3.4 are all mean figures taken from 100 sample problems. A problem consists of a number of jobs (1–14 jobs for each problem) each of which requires a plan and all of which must be scheduled for a specific shop floor. A job is assumed to be one or more identical parts which (usually) remain together as they move through the shop floor. Here each part could have 1–14 processes. A part consists of a blank (the raw material that it is machined from) and a number of features which define its appearance: these can be thought of as describing volumetric removals of material from the blank. A process plan for a given part may be either fixed or flexible: either way the process plan describes the processes that must be carried out (including possible ordering or sequencing constraints) for a specific set of features to appear on the workpiece. However, the process plan does not define the exact way in which that feature is to be machined. The genetic algorithm (GA) searches for near-optimal combinations of processes, machines, tools, and setups (workpiece orientations) for each feature, taking into account interactions with other features and the overall constraints of the problem. In this case the shop floor does not alter between problems. The shop floor consists of 25 machines which vary in the number and diversity of processes that they can carry out. Each process plan is generated from the defined object (including some description of its features and certain possible machining order constraints) and the possible processes that can generate these features on the workpiece; in this case there are one or two applicable processes per feature. For full details see the work of Palmer (1994) and McIlhagga *et al* (1996).

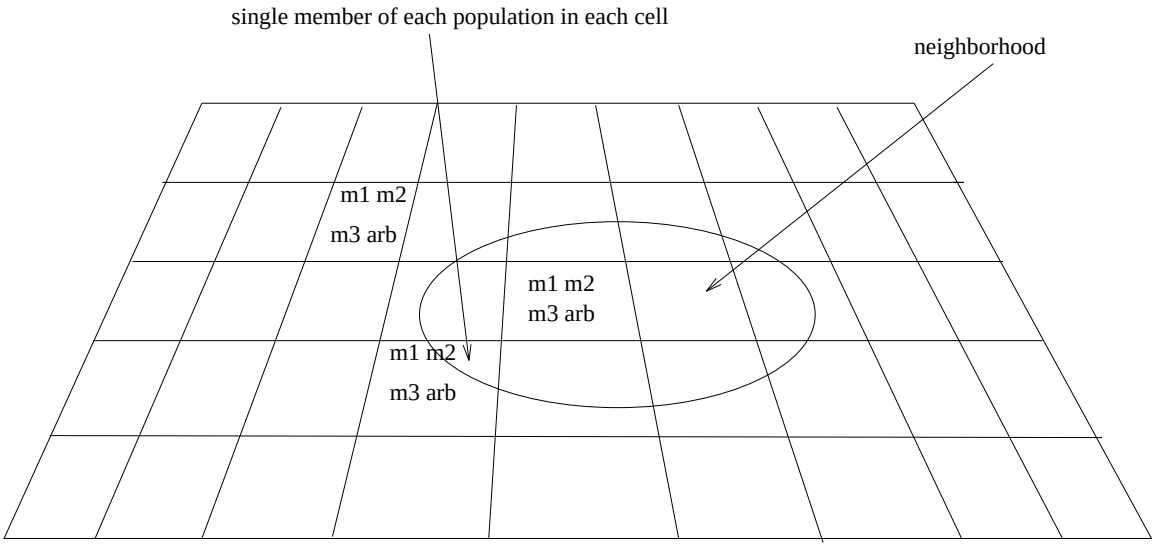


Figure G9.3.4. Distributed interacting populations.

G9.3.2 The design process

G9.3.2.1 Motivation

It is well known that the standard JSS problem is NP-hard (Garey and Johnson 1979). The integrated planning and scheduling problem dealt with here is harder still, involving larger search spaces and more complex constraints, and hence has not attracted much attention until recently. A number of researchers have developed scheduling techniques that allow a small number of options in their process plans (Sycara *et al* 1991, Tonshoff *et al* 1989), but still they are dealing with only a small fraction of the whole problem. Liang and Dutta (1990) have pointed out the need to combine planning and scheduling, but their proposed solution was demonstrated on a very small simplified problem. Given a problem of this complexity it is natural to appeal to stochastic optimization techniques, hence the development of the GA-based method reported here. Comparisons with other techniques are discussed later in section G9.3.4.

G9.3.2.2 The distributed coevolutionary genetic algorithm

A major early concern in this work was how to provide coherent coevolution. The initial, somewhat unsatisfactory, implementation involved a set of interacting standard sequential GAs and is described by Husbands and Mill (1991). A later, more satisfactory implementation, that has been used ever since, spreads each population 'geographically' over *the same* two-dimensional toroidal grid: this is illustrated in figure G9.3.4. Each cell on the grid contains exactly one member of each population. Selection is local: individuals can mate only with those members of their own species in their local neighborhood. Following Hillis (1990) the neighborhood is defined in terms of a Gaussian distribution over distance from the individual; the standard deviation is chosen so as to result in a small number of individuals per neighborhood. Neighborhoods overlap allowing information flow through the whole population without the need for global control. Selection works by using a simple *ranking scheme* within a neighborhood: the most fit individual is twice as likely to be selected as the median individual. Offspring produced replace individuals from their parents' neighborhood. Replacement is probabilistic, using the inverse scheme to selection. In this way genetic material remains spatially local and a robust and coherent coevolution (particularly between arbitrators and process plan organisms) is allowed to unfold. Interactions are also local: costing involves the simulation of the concurrent execution of all the plans *at the same location on the grid* (there will be one for each component, and an arbitrator to resolve conflicts). This implementation consistently gives better results in fewer evaluations than the first. For full details see Husbands (1993, 1994).

C2.4

The overall algorithm is quite straightforward. It can be implemented sequentially or in a parallel asynchronous manner, depending on available hardware.

Overall()

(i) Randomly generate each population, put one member of each population in each cell of a toroidal grid.

(ii) Cost each member of each plan population (phase 1 + phase 2 costs). Phase 1 costs are those intrinsic to a given plan (basic machining costs). Phase 2 costs include waiting times and are calculated by simulating the concurrent execution of all plans represented in a given cell on grid; any resource conflicts are resolved by arbitrator in that cell. Cost arbitrators according to how well conflicts resolved.

(iii) $i \leftarrow 0$.

(iv) Pick random starting cell on the toroidal grid.

(v) Breed each of the representatives of the different populations found in that cell.

(vi) If all cells on the grid have been visited Go to (vii). Else move to next cell, Go to (v).

(vii) If $i <$ MaxIterations, $i \leftarrow i + 1$, Go to (iv). Else Go to (viii).

(viii) Exit.

The breeding algorithm, which is applied in turn to the members of the different populations, is a little more complicated.

Breed(current_cell, current_population)

(i) $i \leftarrow 0$.

(ii) Clear NeighborArray

(iii) Pick a cell in neighborhood of current_cell by generating x and y distances (from current_cell) according to a binomial approximation to a Gaussian distribution. The sign of the distance (up or down, left or right) is chosen randomly (50/50).

(iv) If the cell chosen is not in NeighborArray, put it in NeighborArray, $i \leftarrow i + 1$, Go to (v). Else Go to (iii).

(v) If $i <$ LocalSelectionSize, Go to (iii). Else Go to (vi).

(vi) Rank (sort) the members of current_population located in the cells recorded in NeighborArray according to their cost. Choose one of these using a linear selection function.

(vii) Produce offspring using the individual chosen in (vi) and current_population member in current_cell as the parents.

(viii) Choose a cell from ranked NeighborArray according to an inverse linear selection function. Replace member of current_population in this cell with offspring produced in (vii).

(ix) Find phase one (local) costs for this new individual (not necessary for arbitrators).

(x) Calculate new phase two costs for all individuals in the cell the new individual has been placed in, by simulating their concurrent execution. Update costs accordingly.

(xi) Exit.

The binomial approximation to a Gaussian distribution used in step (iii) falls off sharply for distances greater than two cells, and is truncated to zero for distances greater than four cells.

G9.3.2.3 Requirements

The architecture of the evolutionary systems is such that the evaluation functions can easily be changed to meet the particular requirements of a specific application of the general model. However, the overall requirements will always be the same: minimize the cost of the manufacturing plan for each component (according to particular criteria chosen, e.g. machining and setup costs) and *at the same time* minimize some higher-level criteria such as makespan, mean flowtime, total tardiness, or some combination of these (French 1982).

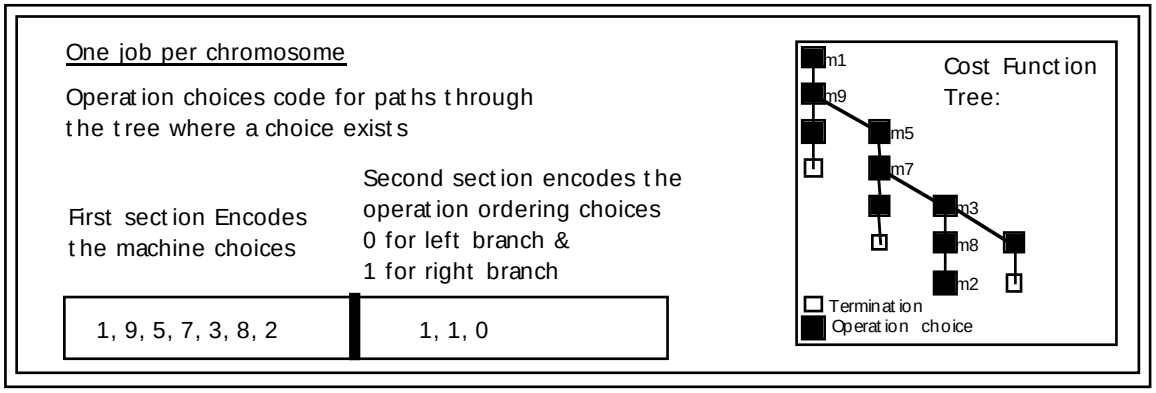


Figure G9.3.5. Process plan encoding.

G9.3.2.4 Representation

As already mentioned, there have been a number of applications of the ecosystem model to different integrated manufacturing planning problems. Each of these has used the same encoding scheme for the arbitrators, but the process plan encodings have been tailored to the particular instance of the integrated problem. The encoding scheme used in the case study reported here will be the only one described in this paper; for a more complex encoding used for a very general version of the problem see the article by Husbands (1993).

For this instance of the problem the process plan chromosomes are divided into two sections: the first part deals with method (i.e. machine) choices, the second with sequence (or ordering) choices (see figure G9.3.5). Method choices are only denoted for jobs where there is more than one applicable method. Currently, all methods have two options and are therefore represented as bits in a bitstring. Lookup tables in the cost function translate these binary values into a machine choice. The method choices are held on the genome in an order which maps on to a set of known operations (1–N), which can be considered the default sequence. For each job, the cost function (see later) maintains a data tree containing the space of legal sequences of operations. Sequence choices on the chromosome are interpreted as routes down the sequence tree for a particular job. The default sequence is always legal, so in cases where the problem description constrains the genome to only one legal sequence, the sequencing information is implicit. The evaluation function is a set of 'data abstraction' routines that traverse a given tree structure, following a route taken as argument, which return with a necessarily valid operation sequence.

The arbitrators are required to resolve conflicts arising when members of the other populations demand the same resources during overlapping time intervals. The arbitrators' genotype is a bitstring which encodes a table indicating which population should have precedence at any particular stage of the execution of a plan, should a conflict over a shared resource occur. A conflict at stage L between populations K and J is resolved by looking up the appropriate entry in the Lth table. Since population members cannot conflict with themselves, and we only need a single entry for each possible population *pairing*, the table at each stage only needs to be of size $N(N-1)/2$, where N is the number of separate component populations. As the arbitrators represent such a set of tables flattened out into a string, their genome is a bitstring of length $SN(N-1)/2$, where S is the maximum possible number of stages in a plan. Each bit is uniquely identified with a particular population pairing and is interpreted according to the function

$$f(n_1, n_2, k) = g\left[\frac{kN(N-1)}{2} + n_1(N-1) - \frac{n_1(n_1+1)}{2} + n_2 - 1\right] \qquad \text{(G9.3.5)}$$

where n_1 and n_2 are unique labels for particular populations, $n_1 < n_2$, k refers to the stage of the plan and $g[i]$ refers to the value of the ith gene on the arbitrator genome. If $f(n_1, n_2, k) = 1$ then n_1 dominates; else n_2 dominates. By using pairwise filtering the arbitrator can be used to resolve conflicts between any number of different species.

G9.3.2.5 Evaluation functions

Each job, j, has the following data associated with it: release date r_j; due date d_j; completion time C_j; flowtime $F_j = C_j - r_j$; lateness $L_j = C_j - d_j$; tardiness $T_j = \max(0, L_j)$; processing time of job j on machine i, P_{ij}.

From this data the following kinds of cost function can be calculated in a straightforward manner: the makespan, $C_{\max}$; the mean flowtime, $\frac{1}{N}\sum_{j=1}^{N} F_j$; the total tardiness, $\sum_{j=1}^{N} T_j$; and the proportion of tardy jobs.

A number of different evaluation functions were experimented with. Particularly good results were obtained with the objective function, O, shown in equation (G9.3.6). This function is to be minimized.

$$O = \frac{1}{N}\sum_{j=1}^{N} F_j + 2\sum_{j=1}^{N} T_j. \tag{G9.3.6}$$

This function, mean flowtime plus twice the total tardiness, is applied to each member of each cell on the two-dimensional grid, including the arbitrators. The flowtime term encourages individually efficient plans and the tardiness term encourages minimal interactions between the plans.

G9.3.3 Development and implementation

The system was developed in C under Unix running on Sun workstations. The distributed coevolutionary GA makes use of the MPI parallel message passing interface protocol, allowing it to run on single workstations, networks of workstations, and specialized parallel machines.

G9.3.4 Results

This section presents results from runs on 100 problems generated from data provided by Palmer (1994). Table G9.3.1 gives the values for various criteria averaged over the 100 problems. The coevolutionary distributed GA (CDGA) results are shown alongside those previously found by Palmer with *simulated annealing* (SA) and local dispatching rule heuristics (K&C). D3.5

Table G9.3.1. A problem set comparison.

Algorithm	Makespan	Proportion tardy	Total tardiness	Total machining time	Mean flowtime
CDGA	81.22	0.14	5.84	171.75	34.86
SA	89.09	0.18	8.87	191.22	36.10
K&C	95.96	0.31	30.28	218.13	41.37

As can be seen from table G9.3.1 the CDGA outperforms SA and K&C on all of the optimization criteria. The mean improvement over SA, averaged over all of the optimization criteria, is 16.58%. The mean improvement over K&C, averaged over all of the optimization criteria, is 37.60%. Each of the methods was run for a comparable number of evaluation function calls.

G9.3.5 Conclusions

In this case study of a complex manufacturing planning problem, we found that for each of a wide range of optimization criteria the ecosystem model consistently outperformed SA and a dispatching rule algorithm. Unlike any of the other techniques, the CDGA produces a number of unique (and quite different) high-quality solutions to the problem on each run. Typically the CDGA would generate eight or nine unique very high-quality solutions to a given problem on a single run. This work has involved adapting Husbands' coevolutionary model of integrated production planning for use with a new set of problems and with cost functions different to those used previously (Husbands 1993). This adaptation turned out to be relatively straightforward, an experience that supports the claim that the coevolutionary model is very general (Husbands 1993).

Acknowledgement

This work was supported by EPSRC grant GR/J40812.

References

Balas E 1969 Machine sequencing via disjunctive graphs: an implicit enumeration algorithm *Operations Res.* **17** 941–57

Carlier J and Pinson E 1989 An algorithm for solving the job-shop problem *Management Sci.* **35** 164–76

French S 1982 *Sequencing and Scheduling: an Introduction to the Mathematics of the Job-Shop* (Chichester: Ellis Horwood)

Garey M and Johnson D 1979 *Computers and Intractability: a Guide to the Theory of NP-Completeness* (San Francisco: Freeman)

Garey M, Johnson D and Sethi R 1976 Complexity of flowshop and jobshop scheduling *Math. Operations Res.* **1**

Graves S 1981 A review of production scheduling *Operations Res.* **29** 646–67

Hillis W D 1990 Co-evolving parasites improve simulated evolution as an optimization procedure *Physica* **42D** 228–34

Husbands P 1993 An ecosystems model for integrated production planning *Int. J. Comput. Integrated Manufacturing* **6** 74–86

——1994 Distributed coevolutionary genetic algorithms for multi-criteria and multi-constraint optimisation *Evolutionary Computing (AISB Workshop, Leeds, 1994, Selected Papers) (Lecture Notes in Computer Science 865)* ed T Fogarty (Berlin: Springer) pp 150–65

Husbands P and Mill F 1991 Simulated co-evolution as the mechanism for emergent planning and scheduling *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, July 1991)* ed R Belew and L Booker (San Mateo, CA: Morgan Kaufmann) pp 264–70

Liang M and Dutta S 1990 A mixed-integer programming approach to the machine loading and process planning problem in a process layout environment *Int. J. Production Res.* **28** 1471–84

McIlhagga M, Husbands P and Ives R 1996 A comparison of optimization techniques for integrated manufacturing planning and scheduling *Parallel Problem Solving from Nature—PPSN IV (Lecture Notes in Computer Science 1141)* ed H-M Voigt, W Ebeling, I Rechenberg and H-P Schwefel (Berlin: Springer) pp 604–13

Muth J and Thompson G 1963 *Industrial Scheduling* (Englewood Cliffs, NJ: Prentice-Hall)

Ow P and Smith S 1988 Viewing scheduling as an opportunistic problem solving process *Ann. Operations Res.* **12**

Palmer G 1994 *An Integrated Approach to Manufacturing Planning* PhD Thesis, School of Engineering, University of Huddersfield

Sycara K and Roth S and Fox M 1991 Resource allocation in distributed factory scheduling *IEEE Expert* **Feb** 29–40

Tonshoff H and Beckendorff U and Anders N 1989 FLEXPLAN—a concept for intelligent process planning and scheduling *CIRP Int. Workshop on CAPP (Hanover, 1989)*

G9.4 An evolutionary approach to airline crew scheduling

David Levine

Abstract

We discuss the application of a parallel implementation of a hybrid genetic algorithm (GA) to the airline crew scheduling problem. Tests on 40 real-world problems were carried out on an IBM SP parallel computer. The algorithm was able to solve to optimality all but one of the small and medium-sized problems, and found good solutions for half of the larger problems. Two limitations were identified: (i) difficulties solving problems with many constraints, and (ii) cases where the penalty term was not strong enough to lead the GA to feasible solutions.

G9.4.1 Introduction

In the airline crew scheduling problem, a set of flight legs (a takeoff and landing) must be flown. The set of flight legs defines an airline's flight schedule (typically on a daily, weekly, or monthly basis) for a particular aircraft type (e.g. Boeing 747). Combinations of flight legs are grouped into round-trip *pairings* that begin and end at a flight crew base. The pairings are defined to meet union and company rules and Federal Aviation Administration requirements. Associated with each pairing, if that pairing is flown, is a cost that reflects salaries, hotel costs, *per diem* expenses, and so on.

The goal of the airline crew scheduling problem is to select a set of pairings so that each flight leg has exactly one crew assigned to it and the total cost is minimized. This problem may be formulated mathematically as the set partitioning problem (SPP):

$$\text{minimize } z = \sum_{j=1}^{n} c_j x_j \tag{G9.4.1}$$

subject to

$$\sum_{j=1}^{n} a_{ij} x_j = 1 \qquad \text{for } i = 1, \ldots, m \tag{G9.4.2}$$

$$x_j = 0 \quad \text{or} \quad 1 \qquad \text{for } j = 1, \ldots, n \tag{G9.4.3}$$

where a_{ij} is binary for all i and j, and $c_j > 0$.

As a model for crew scheduling, the constraints given by (G9.4.2) represent the flight legs, each of which must have exactly one crew assigned to it. The variables represent the pairings, a subset of which are to be selected. The cost of each pairing is c_j, and the matrix elements a_{ij} are defined by

$$a_{ij} = \begin{cases} 1 & \text{if flight leg } i \text{ is part of pairing } j \\ 0 & \text{otherwise.} \end{cases} \tag{G9.4.4}$$

Table G9.4.1 is a simple SPP example problem with four constraints and six variables. The row above the first line contains the cost coefficients of each variable (e.g. $c_2 = 25$). The row below the second line contains the indices of the variables. A feasible solution to this problem is to set $x_2 = x_5 = 1$, and the other $x_j = 0$, with $z = 35$. An infeasible solution to this problem is to set $x_1 = x_5 = 1$, and

the other $x_j = 0$. This solution is infeasible because the first constraint is undercovered (no flight crew is assigned to this flight leg), and the second and third constraints are overcovered (more than one flight crew is assigned to each of these flight legs). For this problem, the optimal solution can be determined by inspection: it is to set $x_3 = x_6 = 1$, and the other $x_j = 0$, with $z = 30$.

G9.4.2 Design process

Several factors motivated this work. First, airline crew scheduling is a visible and economically significant problem, with many references in the operations research literature (Arabeyre *et al* 1969, Barutt and Hull 1990, Gershkoff 1989). Estimates of over a billion dollars a year for pilot and flight attendant expenses have been reported (Anbil *et al* 1991, Barutt and Hull 1990). Hence, developing a successful algorithm is of great practical value. Second, most traditional approaches require the solution of the linear programming relaxation of the SPP ($0 \leq x_j \leq 1$), which can be computationally demanding. Since evolutionary methods can work directly with integer solutions, there is no need to solve the linear programming relaxation. Third, the evaluation function is more easily modified to handle additional constraints than would be the case with more traditional methods. Fourth, evolutionary methods maintain a population of possible solutions that may be of value to a crew scheduling practitioner. Finally, evolutionary approaches have natural parallel implementations and can take advantage of the power of modern parallel computers.

G9.4.2.1 General description

Our evolutionary approach was based on a parallel implementation of a hybrid *genetic algorithm* (GA). B1.2 The sequential GA is:

Input: μ, p_c, t_{max}

Output: $\boldsymbol{a}_{best}$

$t \leftarrow 0$;
$P(t) \leftarrow$ initialize(μ);
$\boldsymbol{F}(t) \leftarrow$ evaluate($P(t)$, μ);
while($\iota(P(t), \boldsymbol{F}(t), t_{max}) \neq$ true) **do**
 $\boldsymbol{a}_{random} \leftarrow$ hillclimb($P(t)$);
 $\boldsymbol{a}_1, \boldsymbol{a}_2 \leftarrow$ select($P(t)$);
 sample $u \sim U(0, 1)$;
 if($u < p_c$)
  then $\boldsymbol{a}_{new} \leftarrow$ crossover($\boldsymbol{a}_1, \boldsymbol{a}_2$);
  else $\boldsymbol{a}_{new} \leftarrow$ mutate($\boldsymbol{a}_1$);
 fi
 $\boldsymbol{a}_{worst} \leftarrow$ worst($P(t)$);
 delete($P(t), \boldsymbol{a}_{worst}$);
 while($\boldsymbol{a}_{new} \in P(t)$) **do**
  $\boldsymbol{a}_{new} \leftarrow$ mutate($\boldsymbol{a}_{new}$);
 od
 $P(t+1) \leftarrow P(t) \cup \boldsymbol{a}_{new}$;
 $\boldsymbol{F}(t) \leftarrow$ evaluate($P(t)$, μ);
 $t \leftarrow t + 1$;
od
$\boldsymbol{a}_{best} \leftarrow$ best($P(t)$);
return($\boldsymbol{a}_{best}$);

Here $P(t)$ is the population of strings at generation t. A steady-state GA is used, with one new individual generated each generation. Each generation a random string, $\boldsymbol{a}_{random}$, is selected and a hill climbing heuristic (see section G9.4.2.7) applied to it. Next, two parent strings, $\boldsymbol{a}_1$ and $\boldsymbol{a}_2$, are selected via *binary tournament selection*, C2.3 and a random number is generated to determine whether to apply crossover *or* mutation. If crossover is applied, we create two new offspring and randomly select one, $\boldsymbol{a}_{new}$, to insert in the population. If mutation is applied, we randomly select one of the parent strings and apply mutation to it. In either case, the new string is tested to see whether it is a duplicate of a string already in the

Table G9.4.1. Example SPP problem.

30	25	20	15	10	10		
0	1	1	1	0	0	=	1
1	0	1	0	1	0	=	1
1	0	0	1	1	1	=	1
1	1	0	0	0	1	=	1
1	2	3	4	5	6		

population. If so, mutation is repeatedly applied to the new string until it is unique. Finally, the least-fit string in the population is deleted, $\boldsymbol{a}_{\text{new}}$ is inserted, and the population is reevaluated.

The parallel GA model we used is the *island model genetic algorithm* (IMGA), where a GA population is divided into several subpopulations, each of which is randomly initialized and runs an independent sequential GA on its own subpopulation. Occasionally, fit strings migrate between subpopulations. We selected the best string in a subpopulation to migrate to a neighboring subpopulation every 1000 iterations. The string replaced was selected by holding a binary tournament and replacing the worst string with probability 0.6. The logical topology of the subpopulations was a two-dimensional toroidal mesh. Each subpopulation was of size 100. C6.3

G9.4.2.2 Representation description

A solution to the SPP problem is given by a binary vector $\boldsymbol{x}$, with the interpretation that $x_j = 1$ (0) if bit j is one (zero) in the binary vector. An SPP solution has a natural encoding in a GA. A bit in a GA string is associated with each column j. The bit is one if column j is included in the solution, and zero otherwise.

G9.4.2.3 Fitness function

Three functions were of interest: the SPP objective function, the evaluation function, and the fitness function. It is the SPP objective function, (G9.4.1), that we wish to have the GA minimize. However, the difficulty with using (G9.4.1) directly is that it does not take into account whether a string is feasible. Therefore, we defined an evaluation function that incorporates both a cost term *and* a penalty term. The generic form of our evaluation function is

$$f(\boldsymbol{x}) = c(\boldsymbol{x}) + p(\boldsymbol{x}) \tag{G9.4.5}$$

where f is the evaluation function, $c(\boldsymbol{x})$ is the cost term ((G9.4.1), the SPP objective function), and $p(\boldsymbol{x})$ is a penalty term (see section G9.4.2.6).

G9.4.2.4 Selection

In *binary* tournament selection (Goldberg 1989, Goldberg and Deb 1991) two strings are chosen randomly from the population, and the more fit string is allocated a reproductive trial. In order to generate a new individual, two binary tournaments are held, each of which produces one parent string. These two parent strings then recombine to produce an offspring.

G9.4.2.5 Operators

In our implementation, crossover *or* mutation is applied to generate a new string. A random number is generated. If it is less than the crossover probability, we apply crossover to generate the new string. Otherwise we use mutation to generate the new string. The mutation rate is constant and set to the reciprocal of the string length.

We experimented with two-point and uniform crossover. Uniform crossover does not have the same disrupting effect on long-defining-length schemata that two-point crossover does (Syswerda 1989), and appeared advantageous for SPP problems. However, uniform crossover is computationally expensive, requiring the generation of a random number for each bit in a string. Our empirical comparison of the two crossovers using a χ^2 test with a significance level of 5% showed no significant difference between them (Levine 1994).

G9.4.2.6 Constraints

The SPP is a highly constrained problem. In the general case, just finding a *feasible* solution to the SPP is NP-complete (Nemhauser and Wolsey 1988). It is likely, at least in the initial stages, that many or most strings in the population are infeasible. Therefore, to evaluate a string, we need a method that takes into account the possible infeasibility of a string. Our approach was to incorporate a penalty term, $p(\boldsymbol{x})$, into the evaluation function (G9.4.5). We experimented with two penalty terms.

The first,

$$p(\boldsymbol{x}) = \sum_{i=1}^{m} \lambda_i \Phi_i(\boldsymbol{x}) \tag{G9.4.6}$$

where

$$\Phi_i(\boldsymbol{x}) = \begin{cases} 1 & \text{if constraint } i \text{ is violated} \\ 0 & \text{otherwise} \end{cases} \tag{G9.4.7}$$

depends on the number of violated constraints.

The second,

$$p(\boldsymbol{x}) = \sum_{i=1}^{m} \lambda_i \sum_{j=1}^{n} \left| a_{ij} x_j - 1 \right| \tag{G9.4.8}$$

measures the magnitude of each constraint's violation.

In equations (G9.4.6) and (G9.4.8), λ_i is a scalar weight that penalizes the violation of constraint i. A good choice for λ_i will reflect not just the 'costs' associated with making constraint i feasible but also the impact on other constraints (in)feasible. We know of no method to calculate an optimal value for λ_i, and made the empirical choice of setting λ_i to the largest c_j of the columns that could cover row i. An empirical comparison of the two penalty terms using a χ^2 test showed no significant difference between them (Levine 1994).

G9.4.2.7 Use of domain knowledge and hybrid methods

We used knowledge of the problem structure both during initialization and in developing a hill climbing heuristic.

Initialization. We found it useful to order the SPP matrix into block 'staircase' form (Pierce 1968). A block, B_i, is defined as the set of columns that have their first one in row i. B_i is defined for all rows but may be empty for some. Within B_i the columns are sorted in order of increasing c_j. Table G9.4.2 shows the example problem of table G9.4.1 after it has been sorted into block staircase form.

Ordering the matrix in this manner is helpful in determining feasibility. In any block, at *most* one x_j may be set to one. Therefore, our initialization scheme (randomly) sets at most one x_j per block to one.

Table G9.4.2. Example SPP problem after sorting.

15	20	25	10	30	10		
1	1	1	0	0	0	=	1
0	1	0	1	1	0	=	1
1	0	0	1	1	1	=	1
0	0	1	0	1	1	=	1
4	3	2	5	1	6		

Hill climbing heuristic. To guide the GA toward feasible solutions, we found the development of a hill climbing heuristic helpful. Our heuristic, called ROW, works as follows. Each time it is called, a row is selected randomly. If the row is undercovered, we select a random column from the set of columns that can cover this row and set it to one. If the row is feasible, we set to zero the column that covers this row, and to one the first column found (if any) that also covers this row, but *only* if the change further minimizes (G9.4.5). If the row is overcovered, we randomly select one of the columns that covers this row and set the *other* columns that cover this row to zero.

G9.4.3 Development and implementation

In the course of this work (Levine 1993, 1994) we tested several operator and parameter choices. In most cases we concluded that the different options we compared performed similarly. This was true for penalty terms, crossover operators, crossover probabilities, and selection strategies. Initialization was an exception. We found that the wide sampling of the initial search space provided by random initialization was preferred to methods that generated the initial population by first calculating a 'good' string with a heuristic, and then generating the rest of the population as random variants of that string.

We found the generational replacement GA, even with elitism, was not very successful in finding (even feasible) solutions to small SPP problems. The steady-state GA was more successful at finding feasible solutions, but still had difficulty finding optimal solutions. This situation motivated our development of the ROW heuristic to combine with the steady-state GA.

ROW has parameters that control whether it makes a first- or best-improving change, and also how many iterations it applies. Our experience was that a 'work quicker, not harder' approach was the most successful (i.e. make first-improving changes and apply ROW infrequently).

Our termination criterion was either when the optimal solution was found (for the test problems, the value of the known optimal solution was stored in the program), or when an iteration limit was reached. For the sequential results reported in table G9.4.3, the iteration limit was 10 000. For the parallel results reported in table G9.4.4, this limit was when all subpopulations had performed 100 000 iterations. The primary performance metric was the quality of the solution found.

To implement the GA and ROW heuristic, we developed a program in ANSI C on a Unix workstation. This program formed the basis for the parallel program, which uses the single-program multiple-data programming model (and also the base upon which the PGAPack parallel GA library (Levine 1995) was built). The experiments were performed on an IBM SP parallel computer with 128 nodes, each of which consisted of an IBM RS/6000 model 370 workstation processor, 128 Mbytes of memory, and a 1 Gbyte disk.

G9.4.4 Results

To test the GA we used a subset of 40 problems from the test set used by Hoffman and Padberg (1993) to test their branch-and-cut algorithm for the SPP. These are real set partitioning problems provided by the airline industry. They are listed in tables G9.4.3 and G9.4.4 in order of increasing numbers of columns (in general, problem difficulty increases as the size of the problem increases). The first 30 problems are small to medium sized (a few thousand columns). The last ten problems are significantly larger, with more columns and more constraints (the largest had 43 749 columns). Details on these problems are given in the articles by Hoffman and Padberg (1993), and Levine (1994).

For the results in table G9.4.3, ten independent runs were made for each test problem using a population size of 100. The 'No opt.' and 'No feas.' columns are the number of times the optimal or feasible solution was found. The '% opt.' column is the percentage from optimality of the best feasible solution. The entry is 'O' if the best feasible solution found was optimal, the percentage from optimality if the best feasible solution was suboptimal, or 'X' if no feasible solution was found.

For most of the small and medium-sized problems, the GA almost always found a feasible solution. Optimal solutions were found, on average, about one fifth of the time. For many other problems, the best feasible solution found was within 3% of optimality. For three of the ten larger problems, feasible solutions were also almost always found. The other large problems, however, presented greater difficulties.

The problems where no feasible solution was found were of two types. Several (`aa01`, `aa04`, `aa05`) had a large number of constraints, and the GA was never able to find a feasible solution. For these problems, approximately 10–25% of the constraints were infeasible at the end of a run. For the others, the GA was able to find infeasible strings with lower evaluation function values than the optimal solution and had concentrated its search on those strings. For these problems the penalty term used in the evaluation function was not strong enough, and the GA exploited that fact. The implication is that penalty terms should be designed such that no infeasible solution is ever better than the worst feasible solution (see e.g. Khuri *et al* 1994, Powell and Skolnick 1993).

For the parallel experiments reported in table G9.4.4 each problem was run once using 1, 2, 4, 8, 16, 32, 64, and 128 subpopulations. Each subpopulation was of size 100. Table G9.4.4 shows the percentage from optimality of the best solution found in any of the subpopulations as a function of the number of

Table G9.4.3. Sequential genetic algorithm results. Reprinted from Levine (1996) with kind permission from Elsevier Science Ltd.

Problem name	No opt.	No feas.	% opt.
nw41	10	10	O
nw32	1	10	O
nw40	0	10	0.004
nw08	0	1	0.033
nw15	8	10	O
nw21	1	10	O
nw22	0	10	0.011
nw12	0	10	0.063
nw39	4	10	O
nw20	0	10	0.009
nw23	6	10	O
nw37	5	10	O
nw26	1	10	O
nw10	0	0	X
nw34	5	10	O
nw43	0	10	0.027
nw42	2	10	O
nw28	6	10	O
nw25	1	10	O
nw38	6	10	O
nw27	1	10	O
nw24	1	10	O
nw35	2	10	O
nw36	0	10	0.003
nw29	0	10	0.024
nw30	4	10	O
nw31	3	10	O
nw19	0	10	0.015
nw33	1	10	O
nw09	0	0	X
nw07	1	10	O
nw06	0	5	0.255
aa04	0	0	X
kl01	0	10	0.022
aa05	0	0	X
nw11	0	0	X
aa01	0	0	X
nw18	0	0	X
kl02	0	9	0.059
nw03	0	9	0.087

subpopulations. A blank entry means the test was not made, usually because of a resource limit or an abort.

For the small and medium-sized problems, the GA was able to find the optimal solution to all but one problem. For approximately two thirds of these problems, only four subpopulations were necessary before the optimal solution was found. For the larger problems, good or optimal solutions were found for half of them. As in table G9.4.3, however, for the problems with a large number of constraints and the problems where the penalty was not strong enough no feasible solution was found.

G9.4.5 Conclusions

An island model implementation of a hybrid GA was an effective approach for solving many small and medium-sized real-world SPP problems. For all but one of these test problems, the optimal solution was found. For several of the larger problems, good feasible solutions were found.

Table G9.4.4. Percent from optimality against number of subpopulations.

Problem name	Number of subpopulations							
	1	2	4	8	16	32	64	128
nw41	O	O	O	O	O	O	O	O
nw32	0.0006	O	0.0006	O	O	O	O	O
nw40	O	O	0.0036	O	O	O	O	O
nw08	X	0.0219	O	O	O	O	O	O
nw15	O	O	O	0.0001	4.4285	O	O	O
nw21	0.0037	0.0037	O	O	O	O	O	O
nw22	0.0735	0.0455	0.0252	O	O	O	O	O
nw12	0.1375	0.0912	0.0332	0.0218	0.0094	O	O	0.0246
nw39	0.0425	O	O	O	O	O	O	O
nw20	0.0091	O	O	O	O	O	O	O
nw23	O	O	O	O	0.0006	O	O	O
nw37	O	0.0163	O	O	O	O	O	O
nw26	0.0011	O	O	O	O	O	O	O
nw10	X	X	X	X	X	X	X	X
nw34	0.0203	0.0214	O	O	O	O	O	O
nw43	0.0831	0.0626	0.0350	O	O	O	O	O
nw42	0.2727	0.0229	O	O	O	O	O	O
nw28	0.0469	O	O	O	O	O	O	O
nw25	0.1040	0.1137	O	O	O	O	O	O
nw38	0.0323	O	O	O	O	O	O	O
nw27	0.0818	0.0567	O	0.0039	O	O	O	O
nw24	0.0826	0.0215	O	0.0015	0.0038	O	O	O
nw35	0.0770	O	0.0171	O	O	O	O	O
nw36	0.0038	0.0010	0.0194	0.0010	0.0019	O	O	O
nw29	0.0580	O	O	0.0116	O	O	O	O
nw30	0.1116	O	O	O	O	O	O	O
nw31	0.0069	0.0069	O	O	O	O	O	O
nw19	0.1559	0.1332	0.0715	0.0880	0.0148	O	O	O
nw33	0.0128	O	O	O	O	O	O	O
nw09	0.0398	X	0.0363	0.0231	0.0155	0.0151	0.154	O
nw07	0.3089	O	O	O	O	O	O	O
nw06	2.0755	0.2532	O	0.1779	0.0448	0.0291	O	O
aa04	X	X	X	X	X			
kl01	0.0524	0.0359	0.0368	0.0303	0.0239	0.0184	0.0082	0.0092
aa05	X	X	X		X			
nw11	X	X	X	X	X	X	X	X
aa01	X	X	X	X	X	X		
nw18	X	X	X	X	X	X	X	X
kl02	0.1004	0.1004	0.0502	0.0593	0.0593		0.0410	0.0045
nw03	0.2732	0.1125	0.1371					0.0481

Limitations did arise, however. First, for some problems the penalty term was not strong enough. In these cases, the GA concentrated its search on infeasible strings that had better evaluation function values than a feasible string would have had. This situation was true independent of the penalty term used. A second limitation was the difficulty solving the problems that had many constraints. For these problems the penalty term seemed adequate, but the GA was still unable to find any feasible solution. Finally, comparisons reported by Levine (1994), although not exact, showed the GA was not competitive with the latest operations research approach for solving SPP problems.

Several areas for future improvement remain. One is to use an adaptive mutation rate or a simulated-annealing-like move in the ROW heuristic as a way to maintain diversity in the population. Recently, Chu and Beasley (1995) have shown that preprocessing the constraint matrix to reduce its size, as well as modifications to some of the basic GA components, can lead to improved performance on SPP problems. Finally, many other methods for solving constrained problems with GAs can be applied to the SPP; these warrant further research.

Acknowledgment

This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Computational and Technology Research, US Department of Energy, under Contract W-31-109-Eng-38.

References

Anbil R, Gelman E, Patty B and Tanga R 1991 Recent advances in crew pairing optimization at American Airlines *Interfaces* **21** 62–74

Arabeyre J, Fearnley J, Steiger F and Teather W 1969 The airline crew scheduling problem: a survey *Transport. Sci.* **3** 140–63

Barutt J and Hull T 1990 Airline crew scheduling: supercomputers and algorithms *SIAM News* **23**

Chu P and Beasley J 1995 *A Genetic Algorithm for the Set Partitioning Problem* Technical Report, Imperial College

Gershkoff I 1989 Optimizing flight crew schedules *Interfaces* **19** 29–43

Goldberg D 1989 *Genetic Algorithms in Search, Optimization and Machine Learning* (New York: Addison-Wesley)

Goldberg D and Deb K 1991 A comparative analysis of selection schemes used in genetic algorithms *Foundations of Genetic Algorithms* ed G Rawlins (San Mateo, CA: Morgan Kaufmann) pp 69–93

Hoffman K and Padberg M 1993 Solving airline crew-scheduling problems by branch-and-cut *Management Sci.* **39** 657–82

Khuri S, Bäck T and Heifhotter J 1994 An evolutionary approach to combinatorial optimization problems *Proc. Ann. ACM Computer Science Conf.* ed D Lizmar (Association for Computing Machinery) pp 66–73

Levine D 1993 A genetic algorithm for the set partitioning problem *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 481–7

——1994 *A Parallel Genetic Algorithm for the Set Partitioning Problem* PhD Thesis, Illinois Institute of Technology, Chicago

——1995 `PGAPack` A general-purpose, data-structure-neutral, parallel genetic algorithm library. Available by anonymous `ftp` from `ftp.mcs.anl.gov` in directory `pub/pgapack`, file `pgapack.tar.Z`.

——1996 Application of a hybrid genetic algorithm to airline crew scheduling *Comput. Operat. Res.* **23** 547–58

Nemhauser G and Wolsey L 1988 *Integer and Combinatorial Optimization* (New York: Wiley)

Pierce J 1968 Application of combinatorial programming to a class of all-zero-one integer programming problems *Management Sci.* **15** 191–209

Powell D and Skolnick M 1993 Using genetic algorithms in engineering design optimization with non-linear constraints *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 424–31

Syswerda G 1989 Uniform crossover in genetic algorithms *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, June 1989)* ed J Schaffer (San Mateo, CA: Morgan Kaufmann) pp 2–9

G9.5 Genetic local search for the traveling salesman problem

Emile H L Aarts and Marco G A Verhoeven

Abstract

Genetic local search is a multilevel neighborhood search technique in which classical local search methods are cast into a genetic framework by introducing populations and recombination neighborhoods.

G9.5.1 Introduction

In an instance of the traveling salesman problem (TSP) one is given a set of n cities and a distance d_{ij} between each pair i, j of cities. The problem is to find a permutation $\pi : \{1, \ldots, n\} \rightarrow \{1, \ldots, n\}$ of the n cities that minimizes the quantity

$$\sum_{i=1}^{n-1} d_{\pi(i),\pi(i+1)} + d_{\pi(n),\pi(1)}.$$

In this quantity, $\pi(i)$ denotes the city that is visited at the ith position in the tour and the quantity itself specifies the length of the tour a salesman would make if he visited the cities in the order specified by the permutation.

The TSP is probably the best-known combinatorial optimization problem and it has served as a proving ground for many new algorithmic ideas (see for instance Lawler *et al* 1985). The TSP is NP-hard (Garey and Johnson 1979), and, consequently, it is unlikely that polynomial-time algorithms exist that solve each instance of the problem to optimality. So, roughly speaking, there are two options. Either one requires optimality of solutions, at the risk of very large, possibly impracticable running times, or one strives for more quickly obtainable solutions at the risk of suboptimality. The first option corresponds to optimization, the second one to approximation.

G9.5.1.1 Optimization

Successful optimization algorithms for the TSP are based on enumeration methods using *branch and bound* techniques in combination with sophisticated techniques for generating *cutting planes*. The currently largest instance solved to optimality counts 7397 cities. However, it took Applegate and coworkers (1990) 3–4 years of CPU time on a network of SPARC2-like machines to find this result. Nevertheless, one can safely state that instances with sizes up to a thousand cities currently can be routinely solved.

G9.5.1.2 Approximation

Practice shows that there are many instances of much larger sizes that must be handled. For instance, in printed circuit board design (Litke 1984) and x-ray crystallography (Bland and Shallcross 1989) instances are known with sizes up to several tens of thousands of cities. This has raised the desire for approximation algorithms that can find near-optimal solutions preferably in small running times. The most popular techniques are tour construction heuristics such as *nearest neighbor*, *nearest insertion*, *farthest insertion*,

and *Christofides' algorithm*. Typical running times of these algorithms range from $O(n^2)$ to $O(n^3)$ (Lawler *et al* 1985). Faster algorithms with running times equal to $O(n \log n)$ or $O(n)$ are obtained by using partitioning approaches (see for instance Karp 1977 and Reinelt 1992). Though fast, the effectiveness of these algorithms is moderate, not better than 10% from optimal (Johnson 1990a).

The best-known approaches for finding approximate solutions to the TSP use *local search* algorithms that are based on the exploration of neighborhoods. Well-known examples for the TSP are the 2-exchange, 3-exchange, and the variable-depth search algorithms of Lin and Kernighan (1973), and the currently best-performing implementations can find solutions within a few percent from optimal for instances with as many as a million cities (see section G9.5.2).

G9.5.1.3 Genetic algorithms for the traveling salesman problem

Early attempts to use *genetic algorithms* to find approximate solutions for the TSP closely followed the classical scheme of what Goldberg (1989) calls a simple genetic algorithm and were based on the use of *binary encodings* of tours and *partially matched crossover operators*. The results that were obtained in this way were rather discouraging when compared with standard approximation algorithms for the TSP. For instance, the experiments of Grefenstette and coworkers (1988) led to solutions as far as 25% from the optimum, in the case of a 50-city TSP. Whitley and coworkers (1989) concluded from their research that the use of binary encodings in genetic algorithms for the TSP is disadvantageous, since it requires special repair algorithms in order to have meaningful crossover operators. Michalewicz (1992) reviews a number of more sophisticated tour representations and corresponding crossover operators, but the general conclusion is that genetic algorithms using only mutation and crossover operators cannot find satisfactory solutions for the larger-problem instances of the TSP. More specifically, no results have been reported for instances with more than 500 cities for which tour lengths have been found within one percent from the optimal tour length.

B1.2
C3.2.1

A major leap in performance, however, can be achieved by combining elements of genetic algorithms and local search, which has led to a class of genetic local search algorithms.

G9.5.2 Local search for the traveling salesman problem

Local search is a general approach to hard combinatorial optimization problems that is based on the exploration of neighborhoods. For an overview we refer to the book by Aarts and Lenstra (1997). Here we restrict ourselves to summarizing a few basic properties.

The use of a local search algorithm presupposes the definition of a problem instance and a neighborhood, which can be formulated as follows.

Definition G9.5.1. An instance *of a combinatorial optimization problem is a pair* $(\mathcal{S}, f)$*, where the* solution space $\mathcal{S}$ *is a finite set of all possible solutions and the* cost function f *is a mapping* $f : \mathcal{S} \to \mathbb{R}$*. In the case of minimization the problem is to find a* globally-minimal solution*, that is, find a solution* $i^* \in \mathcal{S}$ *such that* $f(i^*) \leq f(i)$*, for all* $i \in \mathcal{S}$*. For maximization the definition is equivalent.*

The set $\mathcal{S}$ is generally not given explicitly, that is, by a listing of all elements. Usually, one resorts to the use of a compact representation and a polynomial-time algorithm that either can compute any element in $\mathcal{S}$ or can verify that an element belongs to $\mathcal{S}$.

Definition G9.5.2. Let $(\mathcal{S}, f)$ *be an instance of a combinatorial optimization problem. A* neighborhood structure *is a mapping* $\mathcal{N} : \mathcal{S} \to 2^{\mathcal{S}}$*, which defines for each solution in* $\mathcal{S}$ *a subset of solutions in* $\mathcal{S}$*. The set* $\mathcal{N}(i)$ *is called the* neighborhood *of solution* i*, and each* $j \in \mathcal{N}(i)$ *is called a* neighbor *of* i*. We shall assume that* $i \in \mathcal{N}(i)$*, for all* $i \in \mathcal{S}$*. Furthermore,* $\hat{\imath} \in \mathcal{S}$ *is called* locally minimal *with respect to* $\mathcal{N}$ *if* $f(\hat{\imath}) \leq f(j)$, *for all* $j \in \mathcal{N}(\hat{\imath})$. Note that local optimality depends on the neighborhood structure that is used.

Roughly speaking, a local search algorithm starts off with an initial solution and then continually tries to find better solutions by searching neighborhoods. The literature presents a wealth of local search variants of which *simulated annealing* and tabu search are probably the best-known examples, but also certain types of genetic algorithm and discrete state neural networks can be viewed as local search variants. For an overview we refer to the book by Aarts and Lenstra (1997).

D3.5

A basic version of a local search algorithm is *iterative improvement*. This algorithm is schematically outlined in the following pseudocoded scheme:

```
proc Iterative_Improvement (s ∈ S)
  var w : 2^S;
  begin
    w := ∅ ;
    while N(s) \ w ≠ ∅ do
      s' ∈ N(s) \ w ;
      if f(s') ≥ f(s) then w := w ∪ {s'}
      else s := s'; w := ∅ fi
    od /* s is a local minimum of N */
  end
```

Iterative improvement searches the neighborhood of a current solution for a solution with lower cost. If such a solution is found, the current solution is replaced with this solution. Otherwise, the current solution is returned, which is locally optimal as defined above. Given are an instance $(\mathcal{S}, f)$ of a combinatorial optimization problem and a neighborhood structure $\mathcal{N}$.

In the TSP, $\mathcal{S}$ can be chosen as the set of all Hamilton cycles C in the complete, weighted graph K_n, whose vertices correspond to the cities and edge labels $w_{\{i,j\}}$ are given by the distances d_{ij}. The cost function then can be chosen as

$$f(C) = \sum_{\{i,j\} \in C} w_{\{i,j\}} \qquad \text{for all } C \in \mathcal{S}.$$

A well-known class of neighborhoods for the TSP is given by the k-exchange neighborhoods which can be defined as follows. For all $C \in \mathcal{S}$, $\mathcal{N}^{(k)}(C)$ is given by the set of Hamilton cycles in $\mathcal{S}$ that can be obtained by removing k edges from C and replacing them with k other edges from K_n such that again a Hamilton cycle is constructed. Examples are the frequently used 2-exchange and 3-exchange neighborhoods introduced by Lin (1965) and the Or-exchange neighborhood which is a limited 3-exchange neighborhood due to Or (1976). A quite powerful neighborhood is given by the *variable-depth neighborhood* of Lin and Kernighan (1973), in which neighbors are obtained by generating a sequence of modified 2-exchanges starting from a given tour. The length of a sequence depends among others on the given tour, and hence is variable. The modification is obtained by blocking some of the edges in a 2-exchange. In this way, a local optimum for k-exchanges is obtained without exploring all possible exchanges of k edges. The variable-depth neighborhood is more open minded than a fixed sequence of 2-exchanges in that it does not require that all the interim 2-exchanges encountered be favorable themselves. The idea is that a few small steps in the wrong direction may ultimately be redeemed by large steps in the right direction.

The TSP is probably the best-studied problem for local search algorithms, and is the scene of some of their greatest successes. In summary, we have that algorithms based on the 2- and 3-exchange neighborhoods typically achieve within 3–6% of optimal, while the variable-depth search algorithm of Lin and Kernighan (1973) can achieve within 2%. Moreover, modern implementations of these algorithms have exploited data structures and neighborhood pruning techniques (and modern computers) to obtain surprisingly short running times, with even the slowest, Lin–Kernighan, taking less than an hour to handle 1 000 000 cities. The currently best-performing approximation algorithm for the TSP is the *iterated Lin–Kernighan* algorithm of Johnson and co-workers (Johnson 1990, Fredman *et al* 1995), which routinely finds solutions within 0.5% of optimal in a few minutes running time for randomly generated instances with up to 1 000 000 cities, and which runs with a time complexity slightly more than quadratic. Similar results have been reported by Verhoeven *et al* (1995) for real-life instances with up to approximately 100 000 cities. Their algorithm is a parallel version that performs comparable to the iterated Lin–Kernighan algorithm with speedups ranging from 5 to 15 for a 32-processor network. An excellent overview of the developments in this area is given by Johnson and McGeoch (1997).

G9.5.3 Genetic local search for the traveling salesman problem

A straightforward extension of local search would be to run a single local search algorithm a number of times using different start solutions and keeping the best solution found to return eventually as the final

solution. Several authors have investigated this approach, but no major successes have been reported (see for instance Johnson 1990). It is conceivable that multiple independent runs of a local search algorithm generally will not constitute an effective procedure since, loosely speaking, every individual solution has to find its own way to near-optimal regions.

The above approach could be extended by making the runs dependent. This can be done by combining several neighborhoods, that is, restarting the search in one neighborhood with a solution selected from another neighborhood. This so-called *multilevel approach* can be further extended to a genetic local search approach by using concepts from population genetics and evolution theory. To model genetic local search algorithms we need the following definition.

Definition G9.5.3. Let $(\mathcal{S}, f)$ *be an instance of a combinatorial optimization problem, and* m *a positive integer. A* hyperneighborhood structure *is a mapping* $\mathcal{H}_m : \mathcal{S}^m \rightarrow 2^{\mathcal{S}}$ *which defines for each sequence of* m *solutions in* $\mathcal{S}$ *a subset of solutions in* $\mathcal{S}$.

In terms of genetic algorithm jargon we have for $m = 1$ what is called *mutation*. Moreover, in this case definition G9.5.3 is equal to definition G9.5.2. For $m > 1$ we have *recombination* since in this case the hyperneighborhood relates offspring solutions in $\mathcal{S}$ to parent solutions in $\mathcal{S}$. For instance, if $m = 2$, $\mathcal{H}_2$ defines for each pair of parent solutions $i, j \in \mathcal{S}$ a set $\mathcal{H}_2(i, j) \subset \mathcal{S}$. C3.2 C3.3

Genetic local search algorithms now can be described by the pseudocoded scheme shown below, where we are given an instance $(\mathcal{S}, f)$ of a combinatorial optimization problem and two hyperneighborhood structures $\mathcal{H}_1$ and $\mathcal{H}_m$, with $m > 1$. Furthermore, we use a parent population P of size μ and an offspring population P' of size λ.

```
proc Genetic_Local_Search (P ∈ S^l)
  /* λ, μ, m ≥ 1 */
  begin
    for i := 1 to μ do Iterative_Improvement(s_i) ; od
    stop_criterion := false;
    while ¬ stop_criterion do
      P' := ∅;
      for i := 1 to λ do
        M_i ∈ P^m ;/* Mate */
        s_i ∈ H_m(M_i) ; /* Recombine */
        Iterative_Improvement(s_i) ; /* Improve */
        P' := P' ∪ {s_i} ;
      od
      P :∈ (P ∪ P')^μ ; /* Select */
      evaluate stopcriterion ;
    od
  end
```

G9.5.3.1 Filling in the details

The scheme shown above is just a template which requires further refinements in order to design a successful algorithm. We briefly mention a number of options.

Initialization can be simply done by randomly generated initial populations. However, for the TSP, there is a wealth of tour construction heuristics that could be used to make up an initial population of medium quality and it is well known that on the average better results are found if the local search is started with one of the greedy solutions obtained by tour construction (Johnson and McGeoch 1997).

For the *improvement* step one may choose any of the well-known neighborhoods for the TSP such as the 2-exchange, 3-exchange, Or-exchange or variable-depth neighborhoods. Evidently, the improvement does not need to be restricted to the iterative improvement scheme shown on the previous page. Examples are known of truncated iterative improvement (Suh and Van Gucht 1987), and it is also conceivable that the improvement step can be done by simulated annealing or tabu search.

Mating can be done by randomly selecting solutions from the population or by selecting them according to some deterministic preference rule, for instance by matching solutions in the population. Furthermore, one may partition the population into subsets of solutions before the mating takes place.

Evidently, the *recombination* must try to take advantage of the fact that more than one local optimum is available. Below, we discuss three examples of hyperneighborhoods for the TSP which all use two parent solutions, that is, $m = 2$.

In the *path insertion* neighborhood a neighbor is obtained by randomly selecting a subpath of parent 1 of length between 10 and $n/2$, and extending it to a full tour by adding cities according to the following rules. Let c be the last city in the path. Then the next city to be visited is the successor of c in parent 2 if it has not yet been visited, or the successor of c in parent 1 if it has not yet been visited, or the first as yet unvisited city in parent 2, in that order.

In the *nearest-neighbor* neighborhood a neighbor is obtained by randomly selecting a city and extending it to a full tour by adding cities according to the following rules. Let c be the last city in the path. Then the next city to be visited is the closest as yet unvisited successor city in one of the parents, or a random as yet unvisited city, in that order.

In the *common subpath reversal* neighborhood a neighbor is obtained by randomly selecting a pair of directed subpaths, one in each tour, that contain the same cities but have opposite directions, and replacing the longer path with the shorter one. If no such pair exists the neighborhood is empty.

Selection can be done by the randomized scenario of Goldberg (1989) or by simple deterministic ranking. Other, more sophisticated selection rules such as those discussed in Chapter C2 of this handbook might be used, but are considered beyond the scope of our discussion. Furthermore, it matters whether new offspring solutions compete with the parent solutions or simply substitute them. A promising modification of recombination and selection involves the design of a population structure that defines proximity between positions of individuals, resulting in overlapping cliques, called demes. Then recombination and selection is restricted to take place only among the individuals from each deme (see Gorges-Schleuter 1989). C2

Finally, it should be noted that the genetic local search template on page G9.5:4 is a hybrid approach. It more closely follows the lines of a classical local search algorithm based on the continual improvement of a current solution than the evolutionary computing paradigms as they are for instance presented in Part C of this handbook. However, the borders between evolutionary computing and local search are not strict and combining the best of both sides certainly has led to interesting new algorithmic ideas (see also Chapter D3). D3

G9.5.4 Numerical results

Ulder and coworkers (1991) have tested two basic versions of genetic local search algorithms for the TSP. Both algorithms depart from random populations of solutions, the population sizes being variable and dependent on the problem instances. The first algorithm uses the 2-exchange neighborhood in the improvement step, and the second one uses the variable-depth neighborhood. Both algorithms use the path insertion hyperneighborhood for parent solutions that are obtained by a random matching strategy. Selection is done by deterministic ranking. The algorithm stops when either all tours in the current population have the same length or the length of the best tours has not improved within five successive generations.

Ulder and coworkers (1991) compared the performance of their genetic algorithms with that of the corresponding multistart local search algorithms, as well as with simulated annealing (SA) and its deterministic variant known as threshold accepting (TA), which is due to Dueck and Scheuer (1990). Care was taken to have identical data structures and subroutines wherever possible. The numbers in table G9.5.1 are the average relative deviations from the optimal tour length of the tour lengths of the final solutions obtained by applying the algorithms five times each to eight instances from Reinelt's TSP library (Reinelt 1991), ranging from 48 up to 666 cities. For each instance, the different algorithms were allowed about equal amounts of running time, so the study was focused on effectiveness rather than efficiency.

Table G9.5.1 gives the average deviations from the known optimal solutions. Gen1 and Gen2 perform better than their multistart counterparts. Moreover, Gen2 is superior to the other algorithms. The algorithms were run on a VAX 8650 and the running times ranged from a few seconds for the smaller instances up to 4 hours for the largest instance.

G9.5.5 Discussion

Brady (1985) was the first to use the genetic local search scheme, on page G9.5:4, employing the 2-exchange neighborhoods for improvement and the common subpath reversal hyperneighborhood for

Table G9.5.1. Performance comparison of six local-search-based algorithms: average relative deviation from the optimal tour length (%) for eight well-known instances of the TSP.

Instance	SA	TA	Mult1	Mult2	Gen1	Gen2
GRO48	1.89	1.65	1.35	0	0.19	0
TOM57	1.94	2.88	1.34	0	0.50	0
EUR100	2.59	3.41	3.23	0	1.15	0
GRO120	2.94	2.01	4.57	0.08	1.42	0.05
LIN318	2.37	1.27	6.35	0.37	2.02	0.13
GRO442	2.60	1.31	9.29	0.27	3.02	0.19
GRO532	2.77	1.79	8.34	0.37	2.99	0.17
GRO666	2.19	1.70	8.67	1.18	3.45	0.36

SA: Simulated annealing with 2-exchange neighborhoods.
TA: Threshold accepting with 2-exchange neighborhoods.
Mult1: Multistart iterative improvement with 2-exchange neighborhoods.
Mult2: Multistart iterative improvement with variable-depth neighborhoods.
Gen1: Genetic local search with 2-exchange neighborhoods.
Gen2: Genetic local search with variable-depth neighborhoods.

recombination. For a 64-city instance he could find significantly better tours when compared to a multistart approach. Furthermore, increasing the population size improved the quality of the final results. Brady's approach was limited to small instances, since for the larger instances the hyperneighborhoods he used have a large probability of being empty. Suh and van Gucht (1987) used the nearest-neighbor hyperneighborhood and showed that this substantially improved Brady's approach. They reported results for instances up to 200 cities that were comparable to single-run iterative improvement with 3-exchange neighborhoods. In a later study Jog and coworkers (1989) showed that the additional use of Or-exchanges did not provide strikingly better results. Further improvements have been achieved by Mühlenbein and coworkers (1988), who introduced a more sophisticated mating strategy with a bias imposed for shorter tours on top of a population partitioning strategy known as the *island model*. They used the path insertion C6.3 hyperneighborhood for recombination (see also Gorges-Schleuter 1989). They also introduced a reduction technique in the improvement part similar to that used by Lin and Kernighan (1973), in which edges are fixed that appear in both parents that produce the start solution for the improvement step. The results published by Mühlenbein and coworkers (1988) were quite impressive. At that time, their algorithm was the best approximation algorithm tested on the GRO532 instance. They averaged at 0.19% over optimal which is significantly better than the value of 0.94% that was reported for the single-run variable-depth search (Johnson and McGeoch 1997). By using the variable-depth neighborhood in the improvement step, Ulder and coworkers (1991) could improve this result to 0.17% (see also table G9.5.1).

In conclusion one can state that genetic local search certainly has some potential. However, before it can be added to the list of serious approaches to the TSP, it should be tested for much larger problem instances and its performance should be compared to that of the best-known approaches such as the iterated Lin–Kernighan algorithm mentioned in the introduction. Such a comparison is however not a trivial task since much of the performance of the iterated Lin–Kernighan algorithm is obtained from the use of sophisticated data structures, such as two-level trees and segment trees, which are used for representing sequences of neighboring solutions generated by the algorithm of Johnson and McGeoch (1997). To obtain a fair comparison these data structures should also be used in implementations of genetic local search for the TSP. We feel however that this might turn genetic local search into a strong competitor for the TSP, since it should be possible to improve over the limited 4-exchange in the iterated Lin–Kernighan algorithm by using the more powerful recombination techniques of a genetic approach.

References

Aarts E H L and Lenstra J K 1997 *Local Search in Combinatorial Optimization* (Chichester: Wiley)
Applegate D, Bixby R, Chvatal V and Cook W 1990 Private communication

Bland R G and Shallcross D F 1989 Large traveling salesman problems arising from experiments in x-ray crystallography, a preliminary report on computation *Operat. Res. Lett.* **8** 123–33

Brady R M 1985 Optimization strategies gleaned from biological evolution *Nature* **317** 804–6

Dueck G and Scheuer T 1990 Threshold accepting: a general purpose optimization algorithm *J. Comput. Phys.* **90** 161–75

Fredman M L, Johnson D S, McGeoch L A and Ostheimer G 1995 Data structures for traveling salesman problems *J. Algorithms* **18** 432–79

Garey M R and Johnson D S 1979 *Computers and Intractability: a Guide to the Theory of NP-Completeness* (San Francisco, CA: Freeman)

Goldberg D E 1989 *Genetic Algorithms in Search, Optimization and Machine Learning* (Reading, MA: Addison-Wesley)

Gorges-Schleuter M 1989 Asparagos: an asynchronous parallel genetic optimization strategy *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, June 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 422–7

Grefenstette J J, Gopal R, Rosmaita B and van Gucht D 1985 Genetic algorithms for the traveling salesman problem *Proc. 1st Int. Conf. on Genetic Algorithms and their Applications (Pittsburgh, PA, July 1985* ed J J Grefenstette (Hillsdale, NJ: Laurence Erlbaum) pp 160–8

Jog P, Suh J Y and van Gucht D 1989 The effects of population size, heuristic crossover and local improvement on a genetic algorithm for the traveling salesman problem *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, June 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 110–5

Johnson D S 1990 Local optimization and the traveling salesman problem *Springer Lecture Notes in Computer Science* vol 447 (Berlin: Springer) pp 446–61

Johnson D S and McGeoch L A 1997 The traveling salesman problem: a case study in local optimization *Local Search in Combinatorial Optimization* ed E H L Aarts and J K Lenstra (Chichester: Wiley)

Karp R M 1977 Probabilistic analysis of partitioning algorithms for the traveling salesman in the plane *Math. Operat. Res.* **2** 209–24

Lawler E L, Lenstra J K, Rinnooy Kan A H G and Shmoys D B 1985 *The Traveling Salesman Problem: a Guided Tour of Combinatorial Optimization* (Chichester: Wiley)

Lin S 1965 Computer solutions of the traveling salesman problem *Bell Syst. Tech. J.* **44** 2245–69

Lin S and Kernighan B W 1973 An effective heuristic algorithm for the traveling salesman problem *Operat. Res.* **21** 498–516

Litke J D 1984 An improved solution to the traveling salesman problem with thousands of nodes *Commun. ACM* **27** 1227–36

Michalewicz Z 1992 *Genetic Algorithms + Data Structures = Evolutionary Programs* (Berlin: Springer)

Mühlenbein H, Gorges-Schleuter M and Krämer O 1988 Evolution algorithms in combinatorial optimization *Parallel Comput.* **7** 65–85

Or I 1976 *Traveling Salesman-type Combinatorial Problems and Their Relation to the Logistics of Regional Blood Banking* Phd Thesis, Department of Industrial Engineering and Management Sciences, Northwestern University, Evanston

Reinelt G 1991 Tsplib—a traveling salesman problem library *ORSA J. Comput.* **3** 376–84

——1992 Fast heuristics for large geometric traveling salesman problems *ORSA J. Comput.* **4** 206–23

Suh J Y and van Gucht D 1987 Incorporating heuristic information into genetic search *Proc. 2nd Int. Conf. on Genetic Algorithms (Cambridge, MA, July 1987)* ed J J Grefenstette (Hillsdale, NJ: Laurence Erlbaum) pp 100–7

Ulder N L J, Aarts E H L, Bandelt H-J, van Laarhoven P J M and Pesch E 1991 Genetic local search for the traveling salesman problem *Springer Lecture Notes in Computer Science* vol 496 (Berlin: Springer) pp 109–16

Verhoeven M G A, Aarts E H L and Swinkels P C J 1995 *Parallel Local Search and the Traveling Salesman Problem* manuscript, Philips Research Laboratories, Eindhoven

Whitley D, Starkweather T and Fuquay D 1989 Scheduling problems and traveling salesman: the genetic edge recombination operation *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, June 1989)* ed J Schaffer (San Mateo: Morgan Kaufmann) pp 133–40

G9.6 The set covering problem

J E Beasley

Abstract

In this case study we consider the set covering problem, the classical operations research problem of covering the rows of a zero-one matrix by a subset of the columns at minimum cost. We outline a genetic-algorithm-based heuristic for the problem. The key features of this genetic algorithm are a binary representation, drawn naturally from a zero-one formulation of the problem, a new crossover operator (fusion), and a variable mutation schedule based upon the convergence of the algorithm without mutation and a heuristic operator to ensure feasibility (i.e. to ensure that each individual satisfies the problem constraints). We discuss computational results on a set of standard test problems drawn from the literature. These show that the genetic-algorithm-based heuristic is capable of finding better quality solutions than other approaches currently existing in the literature within a reasonable computation time.

G9.6.1 Introduction

Consider an m-row, n-column, zero-one matrix (a_{ij}), that is, all the elements of the matrix are either zero or one. If $a_{ij} = 1$ we say that column j covers row i, else ($a_{ij} = 0$) column j does not cover row i. Let every column j of this matrix have an associated cost c_j (> 0, $j = 1, \ldots, n$). The set covering problem (SCP) is the problem of choosing a subset of the columns so as to cover all the rows of the matrix at minimum total cost. In terms of mathematics we have that:

defining:

$$x_j = \begin{cases} 1 & \text{if column } j \text{ (cost } c_j > 0) \text{ is in the solution} \\ 0 & \text{otherwise} \end{cases}$$

the SCP is:

$$\text{minimize} \qquad \sum_{j=1}^{n} c_j x_j \tag{G9.6.1}$$

$$\text{subject to} \qquad \sum_{j=1}^{n} a_{ij} x_j \geq 1 \qquad i = 1, \ldots, m \tag{G9.6.2}$$

$$x_j \in \{0, 1\} \qquad j = 1, \ldots, n. \tag{G9.6.3}$$

Equation (G9.6.2) ensures that each row is covered by at least one column chosen to be in the solution and (G9.6.3) is the integrality constraint (a column is chosen or not).

To clarify the problem for those not already familiar with it consider the following example SCP (with $m = 3$ rows and $n = 4$ columns) defined by

$$(c_j) = (2, 3, 4, 4)$$

$$(a_{ij}) = \begin{vmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \end{vmatrix}.$$

Then in terms of equations this SCP is

$$\begin{aligned} \text{minimize} \quad & 2x_1 + 3x_2 + 4x_3 + 4x_4 \\ \text{subject to} \quad & x_1 + x_3 \geq 1 \\ & x_1 + x_4 \geq 1 \\ & x_2 + x_3 + x_4 \geq 1 \\ & x_j \in \{0, 1\} \qquad j = 1, \ldots, 4. \end{aligned}$$

For those readers who find words easier than mathematics, simply think of this SCP as choosing a subset of the four columns which cover all of the three rows at minimum cost. Note that:

(a) $x_1 = 0$, $x_2 = 1$, $x_3 = 1$ and $x_4 = 0$, for example, is not feasible (does not satisfy the constraints) since columns 2 and 3 do not cover row 2.
(b) $x_1 = 1$, $x_2 = 0$, $x_3 = 1$ and $x_4 = 0$, for example, is feasible (does satisfy the constraints) and has cost 6. Hence this is a feasible solution, but it may not be the optimal solution (of minimum total cost).
(c) In fact, for this simple example, it is easy to see that the optimal solution is of cost 5 with $x_1 = x_2 = 1$ and $x_3 = x_4 = 0$.

The SCP (and variants of the SCP which are outside the scope of this section) are important practical problems. For example, probably the most important practical application of the problem is in *crew scheduling* (Rubin 1973, Baker *et al* 1979, Marsten *et al* 1979, Baker and Fisher 1981, Marsten and Shepardson 1981, Crainic and Rousseau 1987, Lavoie *et al* 1988, Desrochers and Soumis 1989, Gershkoff 1989, Hoffman and Padberg 1993). G9.4

However, aside from its practical applications, the SCP is an important academic problem. It has a long history of study in operations research (see, for example, Christofides and Korman 1975, Etcheberry 1977, Chvatal 1979, Balas and Ho 1980, Ho 1982, Hochbaum 1982, Vasko and Wilson 1984, Beasley 1987, Vasko and Wolf 1988, Beasley 1990a, Fisher and Kedia 1990, Beasley and Jörnsten 1992, Harche and Thompson 1994).

The SCP is easy to state and understand, both mathematically and in words. However, it is not easy to solve optimally. By this we mean that we (currently) do not know of an algorithm with a mathematical guarantee of finding the optimal, minimum cost, solution to every possible set covering problem within a time polynomially bounded by the size of the problem. This issue is bound up with the theory of computational complexity, also outside the scope of this section, but it suffices to say here that the problem is NP-complete. For more on computational complexity, see, for example, Garey and Johnson (1979), Karp (1986) or Rayward-Smith (1986).

G9.6.2 Project overview

The set covering problem is a problem that the author has worked on over the years (Beasley 1987, 1990a, Beasley and Jörnsten 1992). Whilst optimal algorithms can currently solve (randomly generated) problems with up to 400 rows and 4000 columns there (plainly) exist problems which are much larger than this. For example, one publically available problem (see Beasley (1990b) or email the message *scpinfo* to *o.rlibrary@ic.ac.uk*) based upon an application in the Italian railway system has 4872 rows and 968 672 columns. For problems of such size heuristic algorithms (algorithms designed to produce good quality (near-optimal) solutions within a reasonable computation time) are the only possible solution approach at the current time.

The original motivation behind this work was an attempt to see if ideas which were emerging from the evolutionary computation community (specifically genetic algorithms) could be successfully applied to the set covering problem—a classical operations research problem which to a large extent has been tackled historically with techniques commonly associated with that community. In this section we present an overview account of this work. For a more detailed technical account, see Beasley and Chu (1996).

G9.6.3 Design process

In applying a *genetic algorithm* (GA) to the SCP, a number of important algorithmic design issues must be considered, as outlined below. B1.2

G9.6.3.1 Requirements

A GA for the SCP is a heuristic: that is, it cannot guarantee to find the optimal solution; instead the goal is to produce a good quality (near-optimal) solution within a reasonable computation time.

One implication of this is that in order to compare the GA with other heuristic algorithms for the SCP we need access to a benchmark set of test problems for which:

(a) either the optimal solutions are known from other work; or

(b) the test problems have been considered by other workers in the field.

Without such a benchmark set it is difficult to judge the real quality of solutions being produced by the GA. For the SCP a benchmark set of test problems is available (see Beasley 1990b).

We would note here that this approach of algorithmic comparison being based upon publically available test problems is becoming increasingly common in operations research.

G9.6.3.2 Representation

The natural representation of an individual for the SCP is the *binary representation*, corresponding to a direct mapping of the variables in the zero-one formulation of the problem (equations (G9.6.1)–(G9.6.3)) to bits. C1.2

Hence we have a one-dimensional bit string of length n representing an individual, with a one at the jth bit position meaning that column j is in the solution, and a zero meaning that column j is not in the solution.

To represent this mathematically let $P_p[j]$ be the jth bit of individual p, so that $P_p[j] = 1$ means that $x_j = 1$ in individual p.

G9.6.3.3 Fitness

The natural way of measuring the fitness of an individual is via the objective function (equation (G9.6.1)) value (hence we attempt to minimize fitness in our GA). Mathematically therefore, the fitness $f(p)$ of individual p is given by $f(p) = \sum_{j=1}^{n} c_j P_p[j]$.

G9.6.3.4 Reproductive system

We used *binary tournament selection* for selecting the two parents who were to have children (in fact, two parents have just a single child as discussed below). C2.3

We used a constant population size of 100 and a steady-state (incremental) population replacement scheme (i.e. replace one member of the population at each iteration) rather than a generational replacement scheme. This was because limited computational experience showed that our GA using steady-state replacement produced better results than using generational replacement.

We found it important to prevent children who were identical to a member of the population (i.e. duplicates) from entering the population. Hence in choosing the member of the population to be replaced by a child, we:

(a) do not put the child in the population if it is identical to a member of the population; otherwise

(b) consider the subgroup consisting of all members of the population whose fitness is above the average fitness for the population and replace a randomly chosen member of this subgroup.

G9.6.3.5 Operator: recombination

We developed a recombination (crossover) operator we call *fusion*. The fusion operator produces just a single child from two parents. Recalling that we are dealing with a minimization problem then for two parents P_1 and P_2 the single child C produced by fusion is defined by:

(a) if $P_1[j] = P_2[j]$ then $C[j] = P_1[j]$ $(= P_2[j])$

(b) if $P_1[j] \neq P_2[j]$ then:
$C[j] = P_1[j]$ with probability $q = f(P_2)/(f(P_1) + f(P_2))$
$C[j] = P_2[j]$ with probability $1 - q$.

The logic here is that if a bit has the same value in both parents it is passed to the child. If a bit differs then the parent who is more fit (has a lower fitness value in our GA) has a higher probability of contributing their bit value to the child.

To see this, suppose that $f(P_1) = 30$ and $f(P_2) = 70$, then at any bit where P_1 and P_2 differ the child will have the bit from P_1 with probability $q = f(P_2)/(f(P_1) + f(P_2)) = 70/(30 + 70) = 0.7$ and the bit from P_2 with probability $1 - q = 0.3$. This seems appropriate because, having a minimization problem, we prefer lower fitness values and so wish to give preference to the bit from the parent (P_1) with the lower fitness value.

G9.6.3.6 Operator: mutation

We applied mutation to each child after crossover. We developed a variable mutation schedule based upon how the GA converged without mutation. The idea here is that in the initial stages of the GA the crossover operator is mainly responsible for the search and so the mutation rate can be set to a low value. As the GA proceeds, crossover will become less productive and so the mutation rate should increase. In our computational work we mutated a fixed number of bits using the expression

$$\text{number of bits mutated} = m_f/[1 + \exp(-4m_g(t - m_c)/m_f)]$$

where t is the number of children that have been generated, m_f is the final mutation rate, m_c is the number of children at which a mutation rate of $m_f/2$ is achieved and m_g specifies the gradient of the above expression at $t = m_c$.

Deciding particular values for m_f, m_c and m_g is problem-dependent and based upon a visual inspection of the convergence of the GA without mutation. In the computational results reported below we used $m_f = 10$, $m_c = 200$ and $m_g = 2$ for all problems (irrespective of problem size).

G9.6.3.7 Constraints

Any individual (solution) after crossover and mutation may, or may not, be feasible (satisfy the constraints, equation (G9.6.2)) for the original problem (the SCP). There are two basic approaches to dealing with infeasibility. These are:

(1) to apply a *penalty function* to penalise infeasible solutions; C5.2
(2) to apply a heuristic operator in an attempt to transform an infeasible solution into a feasible solution (whilst some may term this heuristic operator a *repair operator* we prefer not to use this term).

For the SCP we applied a heuristic operator because for this particular problem it is trivial to develop an operator which *guarantees* to transform an infeasible solution into a feasible solution.

To see this, a simple operator which ensures an individual P_p is feasible is: take each row i which is uncovered in P_p (i.e. $\sum_{j=1}^{n} a_{ij} P_p[j] = 0$) in turn and for each such row take *any* column k which covers i (i.e. $a_{ik} = 1$) and set $P_p[k] = 1$.

Obviously we might expect that such a simple operator would not lead to good quality solutions (since we take no account of the cost of a column in choosing it to cover an uncovered row). However, this example does illustrate that we can always ensure that any individual is feasible.

Whilst in the computational results reported below we used a more complicated operator (involving taking account of column costs, see Beasley and Chu (1996)) we would comment that our experience has been that incorporating problem-specific information into the GA through design of an appropriate heuristic operator to ensure (if possible) feasibility is a key decision in terms of GAs for problems such as the SCP.

G9.6.4 Results

The GA-based heuristic was run on a benchmark set of 65 SCPs of sizes up to 1000 rows and 10 000 columns. For 45 of these test problems, the optimal (minimum cost) solution is known from other work. For the remaining 20 test problems, we know the lowest cost heuristic solution found in the literature.

For each test problem we did 10 trials of the GA, each trial being terminated when 100 000 non-duplicate children had been generated. The performance measures recorded were:

(a) final best (minium cost) solution obtained in each trial;

(b) execution time, i.e. the total computation time for each trial;
(c) solution time, i.e. the computation time for the GA to first reach the final best solution in each trial.

We note here that measure (c) is important as our termination criteria of 100 000 non-duplicate children is purely arbitrary. Plainly significant differences between execution time and solution time imply that the termination criteria could be changed (reduced) without loss of solution quality.

Detailed results can be found in Beasley and Chu (1996) but may be summarized as follows:

(1) For the 45 test problems for which the optimal solution is known, the GA found the optimal solution in at least one of the 10 trials for all but one of the 45 test problems;
(2) For the 20 test problems for which optimal solutions are not known, the GA produced results as good (in 13 problems) or better (in 7 problems) than the previous best-known results from the literature in at least one of the 10 trials.
(3) Comparing execution time with solution time for a number of the test problems did indeed indicate that the termination criteria could be changed (reduced) without loss of solution quality. In fact, average solution time per problem (across 10 trials) never exceeded 800 seconds (on an Iris Indigo workstation), even for the largest problems.
(4) Although the fusion operator performed best, the differences between the GA with fusion and the GA with a different crossover operator (one-point, two-point or uniform) were slight.
(5) A surprisingly high percentage of the children produced by the GA were duplicates of a member of the population (up to a figure of over 60% in some cases). Moreover one-point and two-point crossover produced (on average) more duplicate children than uniform crossover or fusion.

G9.6.5 Conclusions

In this section we have given an overview of a GA for the set covering problem. Based upon extensive computational experience we can conclude that our GA heuristic produces, in a reasonable computation time, results as good, or better, than previous heuristics presented in the literature. Indeed for a number of the test problems considered, we were able to improve upon the previous best-known heuristic solution available in the literature.

References

Baker E K, Bodin L D, Finnegan W F and Ponder R J 1979 Efficient heuristic solutions to an airline crew scheduling problem *AIIE Trans.* **11** 79–85

Baker E and Fisher M 1981 Computational results for very large air crew scheduling problems *OMEGA* **9** 613–8

Balas E and Ho A 1980 Set covering algorithms using cutting planes, heuristics, and subgradient optimization: a computational study *Math. Program.* **12** 37–60

Beasley J E 1987 An algorithm for set covering problems *Eur. J. Operat. Res.* **31** 85–93

——1990a A lagrangian heuristic for set-covering problems *Naval Res. Logistics* **37** 151–64

——1990b OR-Library: distributing test problems by electronic mail *J. Operat. Res. Soc.* **41** 1069–72

Beasley J E and Chu P C 1996 A genetic algorithm for the set covering problem *Eur. J. Operat. Res.* in press

Beasley J E and Jörnsten K 1992 Enhancing an algorithm for set covering problems *Eur. J. Operat. Res.* **58** 293–300

Christofides N and Korman S 1975 A computational survey of methods for the set covering problem *Management Sci.* **21** 591–9

Chvatal V 1979 A greedy heuristic for the set-covering problem *Math. Operat. Res.* **4** 233–5

Crainic T G and Rousseau J-M 1987 The column generation principle and the airline crew scheduling problem *INFOR* **25** 136–51

Desrochers M and Soumis F 1989 A column generation approach to the urban transit crew scheduling problem *Transportation Sci.* **23** 1–13

Etcheberry J 1977 The set-covering problem: a new implicit enumeration algorithm *Operat. Res.* **25** 760–72

Fisher M L and Kedia P 1990 Optimal solution of set covering/partitioning problems using dual heuristics *Management Sci.* **36** 674–88

Garey M R and Johnson D S 1979 *Computers and Intractability: a Guide to the Theory of NP-completeness* (San Francisco: Freeman)

Gershkoff I 1989 Optimizing flight crew schedules *Interfaces* **19** (4) 29–43

Harche F and Thompson G L 1994 The column subtraction algorithm: an exact method for solving weighted set covering, packing and partitioning problems *Comput. Operat. Res.* **21** 689–705

Ho A C 1982 Worst case analysis of a class of set covering heuristics *Math. Program.* **23** 170–80

Hochbaum D S 1982 Approximation algorithms for the set covering and vertex cover problems *SIAM J. Comput.* **11** 555–6

Hoffman K L and Padberg M 1993 Solving airline crew scheduling problems by branch-and-cut *Management Sci.* **39** 657–82

Karp R M 1986 Combinatorics, complexity and randomness *Commun. ACM* **29** 98–117

Lavoie S, Minoux M and Odier E 1988 A new approach for crew pairing problems by column generation with an application to air transportation *Eur. J. Operat. Res.* **35** 45–58

Marsten R E, Muller M R and Killion C L 1979 Crew planning at Flying Tiger: a successful application of integer programming *Management Sci.* **25** 1175–83

Marsten R E and Shepardson F 1981 Exact solution of crew scheduling problems using the set partitioning model: recent successful applications *Networks* **11** 165–77

Rayward-Smith V J 1986 *A First Course in Computability* (Oxford: Blackwell Scientific)

Rubin J 1973 A technique for the solution of massive set covering problems with application to airline crew scheduling *Transportation Sci.* **7** 34–48

Vasko F J and Wilson G R 1984 An efficient heuristic for large set covering problems *Naval Res. Logistics Q.* **31** 163–71

Vasko F J and Wolf F E 1988 Solving large set covering problems on a personal computer *Comput. Operat. Res.* **15** 115–21

G9.7 Knapsack problems

Darrell Whitley, V Scott Gordon and A P Willem Böhm

Abstract

Genetic algorithms are applied to various knapsack problems and compared to bounded depth-first search and a form of parallel branch and bound. Two methods of constructing a fitness function are also considered. The genetic algorithms produce poor results compared to the algorithms using bounds; furthermore, parallel branch and bound can potentially execute faster in parallel than fine-grain cellular genetic algorithms.

G9.7.1 Zero–one knapsack problems

The zero–one knapsack problem is defined as follows. Given n objects with positive weights W_i and positive profits P_i, and a knapsack capacity $\boldsymbol{M}$, determine a subset of the objects, represented by a bit vector $\boldsymbol{X}$ of length n, such that

$$\sum_{i=1}^{n} X_i W_i \leq \boldsymbol{M} \qquad \text{and} \qquad \sum_{i=1}^{n} X_i P_i \text{ is maximal.}$$

An example of a real-world problem which can be modeled as a zero–one knapsack problem is the utilization of communication channels. Given a fixed capacity and a surplus of customers that have different-size communication packets that generate differing amounts of revenue, sell access to a subset of customers in a way that maximizes total profit.

The zero–one knapsack problem is nondeterministic polynomial-time (NP) complete, but a different version of this problem, the *fractional knapsack problem* (Cormen *et al* 1990) has a greedy polynomial-time solution. In the fractional knapsack problem one can place a fraction of an object into the knapsack. Thus, to solve the fractional knapsack problem, objects are iteratively placed in the knapsack by picking the next available object in the unselected set with the highest profit–weight ratio. When no more whole objects fit into the knapsack, select a fractional part of that object in the unselected set with the best profit–weight ratio so that the knapsack is exactly full.

For the zero–one knapsack only whole objects can be placed into the knapsack. In this case, a *greedy approximate solution* is found by inserting objects by profit–weight ratio until the knapsack cannot be filled any further. This also yields a lower-bound estimate on the optimal solution. Problems with poor greedy estimates tend to be harder to solve. It is also easy to see that a greedy solution is not always optimal. Consider the case of three objects where the object with the best profit–weight ratio fills 60% of the knapsack and the other two objects each fill exactly 50% of the knapsack: it is now easy to assign profit–weight ratios such that total profit is maximized by picking the two objects that exactly fill the knapsack rather than the single object with the best profit–weight ratio.

As noted, the greedy solution provides a lower bound on the profit which can be obtained for a zero–one knapsack problem. Any competitive solution should be better than the greedy solution. On the other hand, if we treat the zero–one knapsack problem *as if* it were a fractional knapsack problem, we can also obtain a upper bound on the solution. The solution to the zero–one version of the knapsack problem can be no better than the solution to the fractional knapsack problem.

We can also generate upper and lower bounds with respect to partial solutions. Assume a partial solution has been specified. The partial solution fills part of the knapsack and also removes some objects

from consideration. Treat the residual knapsack capacity as a new knapsack and the unselected objects as the objects in the new *residual* knapsack problem. The upper bound on the residual knapsack problem combined with the profit associated with the partial solution provides an upper bound on the potential profit that can be achieved by extending the partial solution. Bounding algorithms use this information to discard partial solutions that have upper bounds inferior to the current best solution, thus pruning the search space.

At first glance, *genetic algorithms* would appear to be relatively well suited to the zero–one knapsack problem. The most straightforward problem representation is a *binary string*. Traditional recombination operators also work with this representation. In practice, however, the existence of good upper and lower bounds makes it possible to solve knapsack problems with great speed using exact methods such as branch-and-bound algorithms. More analytical methods have also been developed that solve very large knapsack problems (e.g. 250 000 objects) to optimality (Babayer *et al* 1996, Martello and Toth 1990). These particular results were obtained for the *integer knapsack problem*. This version of the knapsack problem can be characterized as follows:

B1.2 C1.2

$$\sum_{i=1}^{n} Y_i W_i \leq \boldsymbol{M} \qquad \text{and} \qquad \sum_{i=1}^{n} Y_i P_i \text{ is maximal}$$

where $\boldsymbol{Y}$ is a vector of nonnegative integers; thus, we have multiple copies of the objects being placed in the knapsack. Of course, it is possible to generate even better greedy solutions to this problem than is possible when the problem is the zero–one knapsack problem. This implies even better lower bounds; nevertheless solving 250 000-object problems to optimality is very impressive.

Small problems are particularly well solved by exact methods. For larger problems, branch-and-bound and bounded depth-first methods with pruning outperform the genetic algorithms both for finding optimal solutions and for finding approximate solutions quickly. These simple methods perform much better than genetic algorithms on this class of problem in spite of the existence of a genetic encoding scheme which exploits useful local information. The results highlight the need for a better understanding of which problems are suitable for genetic algorithms and which problems are not.

One argument that is often offered in favor of genetic algorithms over other methods is the potential for *parallel execution*. However, our results also suggest that it is unclear whether fine-grain parallel genetic algorithms offer a greater potential for parallelism than branch-and-bound methods for a small 80-object test case examined in this paper. At the same time, the kind of parallelism that is available is different for the two approaches.

E2.1.6

The following documents one study of the zero–one knapsack problem. Other work includes that of Khuri *et al* (1994).

G9.7.2 The experiments

Random zero–one knapsack problems were generated based on five parameters: number of objects (n), knapsack capacity (k) as a percentage of total weight of the objects, minimum weight and profit of any object (o), range of weight and profit of any object (v) such that $o + v$ represents the maximum weight (profit) of an object, and a random seed (s). We always use $k = 80\%$ and try to adjust o and v to create a small variance in profit–weight ratio. This seems to generate knapsack problems with poor greedy estimates, although it also generates problems for which the greedy approximation tends to approach the global optimum as the problem size increases. The pseudocode used to generate these problems is as follows. The function randvec(n, l, u) produces a vector of n random integers between l and u:

- $\boldsymbol{M} =$ randvec$(n, o, o + v)$;
- $\boldsymbol{P} =$ randvec$(n, o, o + v)$;
- Sort $\boldsymbol{M}$ and $\boldsymbol{P}$ according to P_i/M_i;
- Capacity $= 0.8 \sum_{i=1}^{n} M_i$.

The test cases include a 20-object problem, an 80-object problem, and several 500-object and 1000-object problems. The 20-object problem was built such that only one of the five objects with the highest profit–weight ratio will fit into the knapsack. This problem was used previously in experiments by Böhm and Egan (1992). The 80-, 500-, and 1000-object problems were built using the random problem generator.

G9.7.3 Binary encodings for zero–one knapsack problems

A simple encoding scheme for zero–one knapsack problems is to let each bit represent the inclusion or exclusion of one of the n objects from the knapsack. Thus, a bitstring of length n can be used to represent candidate solutions. Sorting the bit vector by profit–weight ratio is useful because, for larger problem instances, greedy estimates are often fairly close to the global optimum in terms of their bit representations. If the objects are ordered in the bitstring by profit–weight ratio, then the greedy approximation appears as a series of 1 bits followed by a series of 0 bits. Solutions that improve upon the greedy solution typically have 1 bits as a prefix, 0 bits as a suffix, and a small region of mixed 1s and 0s at the location in the bitstring that corresponds to the 1s to 0s transition in the greedy solution. Because most good solutions are relatively similar, strings are less likely to be disrupted by crossover.

```
Capacity = 50
Weight   = [25,40,20,70]
Profit   = [70,20,5,10]
Optimum  = 75 at 1010
Greedy   = 70 at 1000
```

Figure G9.7.1. A four-object knapsack problem.

Consider the four-object knapsack problem shown in figure G9.7.1 and note that the greedy method used here is more simplistic than necessary. A better greedy method would continue to try inserting objects until reaching the end of the string. The simplistic method used here was chosen for consistency with previous studies (Böhm and Egan 1992).

The problem with this representation is that it is possible to generate infeasible solutions. In other words, setting too many bits to 1 might overflow the capacity of the knapsack. In the above example, the string 1011, which could easily appear during the normal course of genetic search, is an infeasible candidate solution. Two methods of handling overflow are considered: (i) allow infeasible strings and assign a penalty to the evaluation and (ii) ignore the overflow bits.

The first method is implemented by assigning a penalty equal to the amount of overflow. Thus in the above example the string 1011 would have the value of $-(115 - 50) = -65$. This is referred to as the *penalty* method of knapsack evaluation. C5.2

The second method is implemented by adding items one at a time by profit–weight ratio, scanning the bitstring left to right, stopping when the knapsack overflows. Then, the last item that was added is removed. In the above example the string 1011 would be effectively the same as string 1010, since the fourth bit which caused the overflow would be ignored. This is referred to as the *partial-scan* method of knapsack evaluation. The partial-scan method has the interesting characteristic that a string of all 1 bits always evaluates to the greedy approximation. This provides an easy way of seeding the greedy approximation into the population, if desired.

The partial-scan method produced the best results for all of the problems except the 20-object problem. The 20-object problem has a global optimum of 445 and a greedy approximation of 275. The 80-object problem has a global optimum of 25 729 and a close greedy approximation of 25 713. The 500- and 1000-object problems were also generated by using the random method. The profit and weight arrays for the 20- and 80-object problems, and for one of the 500-object problems, are shown in figures G9.7.2–G9.7.4.

```
Profit   = [275,268,260,250,230,63,40,31,41,25,18,23,42,21,25,16,4,5,6,7]
Weight   = [ 55, 54, 53, 52, 51,15,10, 8,13, 7, 6, 8,16, 9,12,12,5,6,7,8]
Capacity = 100
Optimum  = 445  at   01000111011000000000
```

Figure G9.7.2. The 20-object knapsack problem.

```
Profit   = [955,863,753,533,969,725,581,733,975,865,859,499,671,505,827,433,
            707,947,983,557,213,681,891,999,899,401,591,521,543,927,545,985,
            653,101,569,823,431,447,695,303,539,685,839,885,517,153,195,419,
            381,571,357,215,313,225,481,327,359,285,161,391,267,239,181,117,
            281,229,313,179,259,291,313,179,125, 91,219,179, 77, 53, 31,  1]
Weight   = [  4, 48, 98, 86,218,174,150,190,256,242,252,156,232,186,308,162,
            268,388,424,254,102,378,500,624,572,274,408,370,400,720,434,802,
            542, 86,490,736,392,416,656,288,548,702,904,990,606,194,276,596,
            550,836,558,360,602,442,970,680,784,630,378,920,692,624,478,334,
            818,678,930,540,796,900,978,628,478,364,980,940,718,958,944,786]
Capacity = 12000
Optimum  = 25729 at
11111111111111111111111111111111110100000000000000000000000000000000000000
```

Figure G9.7.3. The 80-object knapsack problem.

G9.7.4 Exact methods

Böhm and Egan (1992) describe five algorithms for solving zero–one knapsack problems: *divide and conquer*, *dynamic programming*, *bounded depth-first*, *memo functions*, and *branch and bound*. We consider only bounded depth first and branch and bound since they were reportedly the most effective. Horowitz and Sahni (1978) also review most of these methods for the zero–one knapsack problem in a tutorial fashion, with both example problems and pseudocode.

Bounded depth-first and branch-and-bound algorithms explore a search tree, where points in the search tree represent partial solutions. A lower bound for the best total solution from this point is computed by adding objects with decreasing profit–weight ratio until an object exceeds the knapsack capacity. An upper bound is computed by adding a fractional part of the first object that exceeded the knapsack capacity, such that the knapsack is filled to capacity. The *bounded depth-first* algorithm starts out with the greedy estimate. The upper bound on the potential solution of subtrees representing partial solutions can be compared to the lower bound, the current best solution. Subtrees whose upper bound is worst than the lower bound can be pruned away. The *branch-and-bound* algorithm searches the state space breadth first. Subtrees are cut by calculating the upper bound of a partial solution and comparing it to a shared global variable containing the current best lower bound.

We compare performance by executing the bounded depth-first, branch-and-bound, and genetic algorithms on a simulator of the Manchester Dataflow Machine (Gurd *et al* 1987), which provides statistics such as total instructions executed, critical path, and average parallelism.

G9.7.5 Comparative results: small knapsack problems

Gordon and Whitley (1993) reported results for knapsack problems using several genetic algorithm implementations, including a simple genetic algorithm (SGA), Genitor, a simplified CHC algorithm, several parallel *island models*, and a parallel fine-grain *cellular genetic algorithm* (CGA). Although Genitor performed the best overall, additional experiments using smaller population sizes determined that the optimal performance of any of the genetic algorithms on the 80-object knapsack problem was obtained by using the CGA. With a population size of 25, and the partial-scan encoding method, the CGA requires 26 generations (averaged over 30 runs) to solve the 80-object knapsack. The results reported here are therefore based on the CGA.

C6.3, C6.4

Although it is too slow in practice to run the genetic algorithm to optimality on the 80-object knapsack within the dataflow simulator, it is possible to estimate the total number of instructions required for such execution as follows. First, the number of instructions required for a single generation is estimated by running the algorithm for two and three generations, then finding the difference of the reported number of instructions n for each run. The number of instructions n_g for a single generation is approximately $n_3 - n_2$. It follows that the number of instructions n_i for the initialization phase is approximately $n_2 - 2n_g$. The total number of instructions n_k then for a complete run of k generations is $n_i + kn_g$. Similarly, the critical path cp_k can be estimated using critical paths cp_2 and cp_3 reported for runs of two and three generations.

```
Weight   = [101,100,104,108,101,108,110,110,110,101,106,115,115,116,115,103,114,118,
    116,118,114,124,105,105,107,113,117,122,119,107,126,108,107,113,110,101,117,100,
    114,120,104,118,120,129,103,123,117,133,121,117,126,119,133,124,111,133,136,117,
    129,136,100,140,102,140,111,127,135,142,126,105,118,124,143,125,127,134,133,148,
    133,143,103,116,126,124,115,139,139,150,107,121,127,139,153,101,155,145,126,115,
    146,133,149,142,111,152,145,159,156,115,115,143,102,149,103,125,110,124,138,145,
    109,110,124,131,164,129,153,130,131,157,140,145,105,122,129,124,124,126,148,165,
    100,117,163,169,158,167,162,105,111,159,118,143,168,102,126,123,151,132,154,111,
    169,149,115,155,159,107,174,166,168,170,146,172,164,121,169,172,147,170,105,150,
    167,159,109,112,180,113,125,104,107,107,110,136,152,116,129,182,135,180,184,129,
    119,105,151,155,143,176,179,102,103,181,185,114,115,154,170,175,139,165,170,148,
    174,105,178,188,136,138,143,150,174,137,139,193,147,101,165,179,155,187,135,147,
    150,156,156,174,167,157,145,122,193,184,181,169,144,189,189,121,171,167,159,105,
    195,194,122,194,148,110,104,162,187,154,122,144,108,168,116,139,136,178,154,182,
    116,135,168,198,142,192,188,167,167,161,176,155,168,196,174,158,186,135,189,188,
    157,152,113,150,145,179,196,155,199,118,147,166,154,199,158,148,177,192,170,151,
    112,193,136,188,148,173,164,195,192,182,170,193,181,149,141,145,198,182,163,191,
    191,198,186,185,148,190,179,169,173,158,171,143,169,168,120,119,181,140,140,152,
    149,172,142,158,198,157,162,148,176,153,135,191,195,160,171,193,193,181,169,163,
    157,150,134,174,194,198,146,165,185,191,132,146,198,187,147,146,183,164,179,128,
    160,137,146,173,140,147,178,195,158,140,192,183,194,175,132,135,143,180,150,132,
    196,154,186,197,153,146,194,167,159,140,155,181,156,193,185,198,172,150,153,189,
    156,141,184,188,195,193,162,186,148,189,147,188,191,152,176,198,182,172,194,186,
    175,181,173,197,169,193,158,188,198,197,185,157,195,163,160,193,184,167,160,163,
    198,191,196,175,182,176,174,178,173,189,171,196,183,199,196,182,184,195,185,195,
    188,199]
Profit   = [197,187,192,199,186,196,196,194,194,176,182,197,196,194,191,171,188,194,
    188,191,184,199,167,167,170,178,184,188,183,164,193,165,163,172,167,153,177,151,
    172,181,156,177,180,193,154,183,174,197,179,173,186,175,195,181,162,193,197,169,
    186,196,144,198,143,196,155,177,188,196,173,144,161,169,194,169,171,180,178,197,
    176,189,135,152,165,161,149,180,180,194,138,156,162,177,194,128,196,183,159,145,
    184,167,187,178,139,190,181,198,194,143,143,177,126,184,127,154,135,152,169,177,
    133,134,151,159,199,156,185,157,158,189,168,174,126,146,154,148,148,149,175,195,
    118,138,192,199,186,196,190,123,130,186,138,167,196,119,146,142,174,152,177,127,
    193,170,131,176,179,120,195,186,188,190,163,192,183,135,188,191,163,188,116,165,
    183,174,119,122,196,123,136,113,116,116,119,147,164,125,139,196,145,193,197,138,
    127,112,161,165,152,187,190,108,109,191,195,120,121,162,178,183,145,172,177,154,
    181,109,184,194,140,142,147,154,178,140,142,197,150,103,168,182,157,189,136,148,
    151,157,157,173,166,156,144,121,191,182,179,167,142,186,186,119,168,164,156,103,
    191,190,119,189,144,107,101,157,181,149,118,139,104,161,111,133,130,170,147,173,
    110,128,159,187,134,181,177,157,157,151,165,145,157,183,162,147,173,125,175,174,
    145,140,104,138,133,164,179,141,181,107,133,150,139,179,142,133,159,172,152,135,
    100,172,121,167,131,153,145,172,169,160,149,169,158,130,123,126,172,158,141,165,
    165,171,160,159,127,163,153,144,147,134,145,121,143,142,101,100,152,117,117,127,
    124,143,118,131,164,130,134,122,145,126,111,157,160,131,140,158,158,147,137,132,
    127,121,108,140,156,159,117,132,148,152,105,116,157,148,116,115,144,129,140,100,
    125,107,114,135,109,114,138,151,122,108,148,141,149,134,101,103,109,137,114,100,
    148,116,139,147,114,108,143,123,117,103,114,133,114,141,135,144,125,109,111,137,
    113,102,133,135,140,138,115,132,105,134,104,133,135,107,123,138,126,119,133,127,
    119,123,117,133,114,130,106,126,132,131,123,104,129,107,104,124,118,107,102,103,
    122,117,120,107,110,106,104,106,103,112,101,114,106,115,110,102,103,109,103,106,
    102,106]
```

Figure G9.7.4. One of the 500-object problems, having capacity 49 117 and optimum 55 928.

For the CGA with population size 25, the dataflow simulator reports $n_2 = 713\,174$ and $n_3 = 991\,405$. It follows that $n_g = 991\,405 - 713\,174 = 278\,231$, $n_i = 713\,174 - 2(278\,231) = 156\,712$. For critical path, the dataflow simulator reports $cp_2 = 32\,871$ and $cp_3 = 34\,520$. It follows that $cp_g = 34\,520 - 32\,871 = 1649$, $cp_i = 32\,871 - 2(1649) = 29\,573$. Therefore the total number of instructions n_k and the critical path length cp_k for solving the problem are approximately given by

$$n_k = n_i + 26n_g = 156\,712 + 26(278\,231) = 7\,390\,718$$
$$cp_k = cp_i + 26cp_g = 29\,573 + 26(1649) = 42\,874.$$

Table G9.7.1 compares the statistics generated by the dataflow simulator.

Table G9.7.1. The genetic algorithm versus other methods on the 80-object knapsack.

Algorithm	Total instructions	Critical path
Massively parallel GA	7 390 718	42 874
Bounded depth first	1 142 637	355 711
Branch and bound	128 636	16 582

Both bounded depth-first and branch and bound find solutions faster (fewer total instructions) than the genetic algorithm. The short critical path for branch and bound indicates that this algorithm could execute faster in parallel than the genetic algorithm. At the same time, the parallelism of the branch-and-bound approach is irregular and a bottleneck exists in terms of the need to communicate lower-bound information. The genetic algorithm has more regular parallelism, only local communication, and no bottleneck. At the same time, the figure for the critical path of the genetic algorithm may be optimistic, since some of the parallelism is bit level parallelism that may not be realizable in an actual implementation. Thus, the question of which approach is best suited to parallel implementation is unclear. It is nonetheless true that bounded depth first is about six times faster than the genetic algorithm, and branch and bound is about 60 times faster.

G9.7.6 Large knapsack problems

Since knapsack problems define a search space of 2^n combinations of objects, exhaustive search methods will eventually fail on very large problems. While this is probably also true for genetic algorithms, perhaps the genetic algorithm can provide better solution estimates part way through the search than standard methods can provide. To test this, the CGA and the bounded depth-first and branch-and-bound algorithms were run on a set of four larger knapsack problems, two of size 500 and two of size 1000. All of the algorithms were run on a Sparc 2 with 32 Mbytes of memory. The 'best-so-far' values for each algorithm at various times during the search were compared. Several population sizes for the genetic algorithm were tested.

Recall that for larger problems the greedy estimate is close to the global optimum. While one would expect this to benefit the genetic algorithm performance, it also helps simple exact methods prune the search space more effectively. In fact, it was necessary to generate 30 knapsack problems of 500 and 1000 objects in order to find *four* that the exact methods did not solve within one second on the Sparc 2. Performance results are shown in table G9.7.2.

Table G9.7.2. Genetic algorithm against other search methods on hard knapsack problems.

		ks_1^{500}	ks_2^{500}	ks_1^{1000}	ks_2^{1000}
Global optimum		55 928	56 448	336 983	630 972
Greedy estimate		55 907	56 366	336 699	630 397
Depth first	time to solve	1 h	$>$ 3 h	$>$ 3 h	25 min
	estimate @ 10 s	55 927	56 446	336 982	630 972
Branch and bound	time to solve	1 s	7 s	24 s	never[a]
	estimate @ 10 s	solved	solved	—	—
CGA popsize = 25	time to solve	∞	∞	∞	∞
	estimate @ 90 s	55 879	56 349	334 963	626 398
CGA popsize = 100	time to solve	∞	∞	∞	∞
	estimate @ 90 s	55 820	56 315	329 672	616 064
	estimate @ 180 s	55 912	56 419	336 583	630 154
CGA popsize = 400	time to solve	∞	∞	∞	∞
	estimate @ 20 min	55 924	56 437	336 852	630 594

[a] Branch and bound exceeds memory for the ks_2^{1000} problem.

'Best-so-far' values for each algorithm at various times during the search are compared. On 500-object problems, the genetic algorithm requires 3 minutes just to reach the greedy estimate (note that the greedy estimate can always be determined directly, requiring only a sort which can be done in polynomial time). Branch and bound performs significantly faster than either of the other algorithms, but space demands cause it to fail on one of the problems. Bounded depth-first also clearly beats the genetic algorithm, since the genetic algorithm never reaches the estimate which the bounded depth first algorithm finds after 10 seconds. It is interesting to note that on problem ks_2^{1000}, the bounded depth-first algorithm finds the global optimum after only 10 seconds, but requires 25 more minutes to *know* that it is the optimum. *The genetic algorithm takes 25 minutes just to find the greedy estimate*.

The genetic algorithm results can be improved slightly by seeding the population with a copy of the greedy estimate. As stated earlier, this is easily done by setting all of the bits in one of the strings to 1. Tests determined that doing this does not change the comparison with bounded depth first with regards to solving the problems to optimality, or obtaining better estimates. For the longer time periods, the estimates produced by the genetic algorithm were close to those shown in table G9.7.2.

G9.7.7 Conclusions

These findings strengthen the notion that genetic algorithms are general-purpose algorithms *not intended to supplant existing methods for solving all problems*. The algorithms used here are also rather simple general purpose search algorithms. Martello and Toth (1990) report solving much larger knapsack problems than ours in under 1 minute using more specialized forms of branch and bound; code for branch and bound is also readily available (see e.g. Horowitz and Sahni 1978). The analytical methods proposed by Babayev *et al* (1996) appear to be even better. It seems reasonable to infer that pure genetic search could not match the kind of performance these researchers report since the cost of evaluating a population large enough to adequately sample such a huge space would be excessive.

Application domains such as the zero–one knapsack may not be well suited to blind genetic search, and genetic approaches to such problems may instead require a hybrid approach. On the other hand, problem representation can make a huge difference and perhaps a better representation could dramatically improve the performance of the genetic algorithm. Yet, the bounded depth-first and branch-and-bound methods exploit a great deal of problem specific knowledge and run extremely fast. These algorithms also do not have the overhead of a population-based search. We clearly need a better understanding of which problems cannot be solved practically by exact methods, and which may lend themselves instead to genetic or hybrid approaches.

Some knapsack problems are included in the current paper. We would advise researchers interested in working with the knapsack problem to use these problems only as a starting point. Any serious studies of the knapsack problem should look at much larger problems (e.g. 100 000 variables) and should compare results with methods such as branch and bound.

References

Babayev D, Glover F and Ryan J 1996 A new knapsack solution approach by integer equivalent aggregation and consistency determination *Mathematical Programming* at press

Cormen T, Leiserson C and Rivest R 1990 *Introduction to Algorithms* (Cambridge, MA: MIT Press)

Böhm A and Egan G 1992 Five ways to fill your knapsack *Proc. 2nd Sisal Workshop* LLNL CONF-9210270

Gordon V and Whitley D 1993 Serial and parallel genetic algorithms as function optimizers *Proc. 5th Int. Conf on Genetic Algorithms (Urbana-Champaign, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann)

Gurd J, Kirkham C and Böhm W 1987 The Manchester dataflow computing system *Exper. Parallel Comput. Arch. (Special Topics in Supercomputing 1)* ed J Dongarra (Amsterdam: North-Holland)

Horowitz E and Sahni S 1978 *Fundamentals of Computer Algorithms* (Computer Science Press)

Khuri S, Bäck T and Heitkötter F 1994 The zero–one multiple knapsack problem and genetic algorithms *Proc. 1994 ACM Symp. on Applied Computing* ed E Deaton, D Oppenheim, F Urban and H Berghel (ACM)

Martello S and Toth P 1990 *Knapsack Problems: Algorithms and Computer Implementations* (New York: Wiley)

G9.8 The transportation problem

Zbigniew Michalewicz

Abstract

This case study discusses an application of an evolutionary computation technique for the nonlinear transportation problem. The system described, Genetic-2n, is based on the problem-specific representation and feasibility-preserving operators; its performance is compared with a standard optimization tool GAMS (with MINOS optimizer).

G9.8.1 Introduction

Two systems, Genetic-2 and Genetic-2n, have been developed (Vignaux and Michalewicz 1991, Michalewicz *et al* 1991) for linear and nonlinear transportation problems. To the best of the author's knowledge these were the first genetic algorithm-based systems to use nonstring chromosome structures†; specialized 'genetic' operators were introduced to preserve feasibility of solutions. This article describes the transportation problem and one of these systems: Genetic-2n, a nonstandard evolutionary algorithm for the nonlinear transportation problem.

The transportation problem (Taha 1987) is one of the simplest constrained optimization problems that have been studied. It seeks the determination of a minimum-cost transportation plan for a single commodity from a number of sources to a number of destinations. A destination can receive its demand from one or more sources. The objective of the problem is to determine the amount to be shipped from each source to each destination such that the total transportation cost is minimized.

If the transportation cost on a given route is directly proportional to the number of units transported, we have a *linear transportation problem*. Otherwise, we have a *nonlinear transportation problem*.

Assume there are n sources and k destinations. The amount of supply at source i is source(i) and the demand at destination j is dest(j). The cost of transporting flow x_{ij} from source i to destination j is given as a function f_{ij}. Thus the total cost is a separable function of the individual flows rather than interactions between them. The transportation problem is given as

$$\text{minimize total} = \sum_{i=1}^{n} \sum_{j=1}^{k} f_{ij}(x_{ij})$$

subject to

$$\sum_{j=1}^{k} x_{ij} \leq \text{source}(i) \qquad \text{for } i = 1, 2, \ldots, n$$

$$\sum_{i=1}^{n} x_{ij} \geq \text{dest}(j) \qquad \text{for } j = 1, 2, \ldots, k$$

$$x_{ij} \geq 0 \qquad \text{for } i = 1, 2, \ldots, n \text{ and } j = 1, 2, \ldots, k.$$

The first set of constraints stipulates that the sum of the shipments from a source cannot exceed its supply; the second set requires that the sum of the shipments to a destination must satisfy its demand.

† Fogel *et al* (1966) applied evolutionary programming to finite-state machines represented as matrices.

The above problem implies that the total supply $\sum_{i=1}^{k}$ source(i) must at least equal total demand $\sum_{j=1}^{n}$ dest(j). When total supply is equal to total demand (total flow), the resulting formulation is called a *balanced* transportation problem. It differs from the above only in that all constraints are equations; that is,

$$\sum_{j=1}^{k} x_{ij} = \text{source}(i) \qquad \text{for } i = 1, 2, \ldots, n$$

$$\sum_{i=1}^{n} x_{ij} = \text{dest}(j) \qquad \text{for } j = 1, 2, \ldots, k.$$

Vignaux and Michalewicz (1991) described a few evolutionary systems for the linear transportation problem. However, all systems based on a string representation gave disappointing results: they either returned infeasible solutions or they could not be generalized into the nonlinear case (like system Genetic-1; Vignaux and Michalewicz 1991).

Perhaps the most natural representation of a solution for the transportation problem is a two-dimensional structure. After all, this is how the problem is presented and solved by hand. In other words, a matrix $V = (x_{ij})$ $(1 \le i \le k,\ 1 \le j \le n)$ may represent a solution; each x_{ij} is a real number.

In this case study we describe an evolutionary system, Genetic-2n (Michalewicz *et al* 1991), which is based on such a two-dimensional representation for the balanced transportation problem; we describe the most important components of this system in turn.

The *initialization* procedure creates an individual solution which satisfies all constraints. It will also turn out to be a fundamental component of one of the mutation operators. It will be called a sufficient number of times at the start of a run to construct a starting population of solutions.

Input: arrays dest$[k]$, sour$[n]$
Output: an array (x_{ij}) such that $x_{ij} \ge 0$ for all i and j, $\sum_{j=1}^{k} x_{ij} =$ sour$[i]$ for $i = 1, 2, \ldots, n$, and $\sum_{i=1}^{n} x_{ij} =$ dest$[j]$ for $j = 1, 2 \ldots, k$, i.e. all constraints are satisfied.

procedure initialization;
  set all numbers from 1 to kn as unvisited
  while there is an unvisited number **do**
    select an unvisited random number q from 1 to kn and set it as visited
    set (row) $i \leftarrow \lfloor (q-1)/k + 1 \rfloor$
    set (column) $j \leftarrow (q-1)\ mod\ k + 1$
    set val $\leftarrow$ min(sour$[i]$, dest$[j]$)
    set $x_{ij} \leftarrow$ val
    set sour$[i] \leftarrow$ sour$[i] -$ val
    set dest$[j] \leftarrow$ dest$[j] -$ val
  od

Procedure *initialization* creates a matrix of at most $k+n-1$ nonzero elements such that all constraints are satisfied. It is the method used for the linear case of the transportation problem (see Vignaux and Michalewicz 1991). Although other initialization procedures are feasible, this method will generate a solution that is at a vertex of the simplex which describes the convex boundary of the constrained solution space. The following example shows how it works.

Example. Let us consider a matrix with the following constraints:

$$\text{sour}[1] = 15.0,\ \text{sour}[2] = 25.0,\ \text{and sour}[3] = 5.0$$
$$\text{dest}[1] = 5.0,\ \text{dest}[2] = 15.0,\ \text{dest}[3] = 15.0,\ \text{and dest}[4] = 10.0.$$

There are altogether $3 \times 4 = 12$ numbers; all of them are unvisited at the beginning. Select the first random number, say, 10. This translates into row number $i = 3$ and column number $j = 2$. The val $=$ min(sour[3], dest[2]) $= 5.0$, so $v_{32} = 5.0$. After the first iteration, sour[3] $= 0.0$ and dest[2] $= 10.0$.

We repeat these calculations with the next three random (unvisited) numbers, say 8, 5, and 3 (corresponding to row 2 and column 4, to row 2 and column 1, and to row 1 and column 3, respectively).

The resulting matrix (v_{ij}) at this stage has the following contents:

	0.0	**10.0**	**0.0**	**0.0**
0.0			15.0	
10.0	5.0			10.0
0.0		5.0		

Note that the values of sour[i] and dest[j] are those given after four iterations.

If the further sequence of random numbers is 1, 11, 4, 12, 7, 6, 9, 2, the final matrix produced (with an assumed complete sequence of random numbers $\langle 10, 8, 5, 3, 1, 11, 4, 12, 7, 6, 9, 2 \rangle$) is

	0.0	**0.0**	**0.0**	**0.0**
0.0	0.0	0.0	15.0	0.0
0.0	5.0	10.0	0.0	10.0
0.0	0.0	5.0	0.0	0.0

Obviously, after 12 iterations all sour[i] and dest[j] are equal to 0.0. Note also that there are other sequences of numbers for which procedure *initialization* would produce the same solution.

We defined three 'genetic' operators: two mutations and one crossover.

G9.8.2 Mutation

There are two types of mutation. The first mutation introduces as many zero entries into the matrix as possible. The second is modified to avoid choosing zero entries by selecting values from a range. Each will be discussed in turn.

Mutation-1. Assume that $\{i_1, i_2, \ldots, i_p\}$ is a subset of $\{1, 2, \ldots, k\}$, and $\{j_1, j_2, \ldots, j_q\}$ is a subset of $\{1, 2, \ldots, n\}$ such that $2 \leq p \leq k$, $2 \leq q \leq n$.

Denote a parent for mutation by the $(k \times n)$ matrix $V = (x_{ij})$. Then we can create a $(p \times q)$ submatrix $W = (w_{ij})$ from all elements of the matrix V in the following way: an element $x_{ij} \in V$ is in W if and only if $i \in \{i_1, i_2, \ldots, i_p\}$ and $j \in \{j_1, j_2, \ldots, j_q\}$ (if $i = i_r$ and $j = j_s$, then the element x_{ij} is placed in the rth row and sth column of the matrix W).

Now we can assign new values $\text{sour}_W[i]$ and $\text{dest}_W[j]$ $(1 \leq i \leq p,\ 1 \leq j \leq q)$ for matrix W:

$$\text{sour}_W[i] = \sum_{j \in \{j_1, j_2, \ldots, j_q\}} x_{ij} \qquad 1 \leq i \leq p$$

$$\text{dest}_W[j] = \sum_{i \in \{i_1, i_2, \ldots, i_p\}} x_{ij} \qquad 1 \leq j \leq q.$$

We can use the procedure *initialization* to assign new values to the matrix W so that all constraints $\text{sour}_W[i]$ and $\text{dest}_W[j]$ are satisfied. Then we replace corresponding elements of matrix V by new elements from the matrix W. In this way all the global constraints (sour[i] and dest[j]) are preserved.

The following example will illustrate the mutation operator.

Example G9.8.1. A transportation problem is defined with four sources and five destinations. The constraints are

$$\text{sour}[1] = 8.0,\ \text{sour}[2] = 4.0,\ \text{sour}[3] = 12.0,\ \text{sour}[4] = 6.0$$
$$\text{dest}[1] = 3.0,\ \text{dest}[2] = 5.0,\ \text{dest}[3] = 10.0,\ \text{dest}[4] = 7.0,\ \text{dest}[5] = 5.0.$$

Assume that the following matrix V was selected as a parent for mutation:

0.0	0.0	5.0	0.0	3.0
0.0	**4.0**	**0.0**	0.0	**0.0**
0.0	0.0	5.0	7.0	0.0
3.0	**1.0**	**0.0**	0.0	**2.0**

Suppose that two rows $\{2, 4\}$ and three columns $\{2, 3, 5\}$ are selected (typed in boldface in the matrix above). Then the corresponding submatrix W is

4.0	0.0	0.0
1.0	0.0	2.0

Note that $\text{sour}_W[1] = 4.0$, $\text{sour}_W[2] = 3.0$, $\text{dest}_W[1] = 5.0$, $\text{dest}_W[2] = 0.0$, $\text{dest}_W[3] = 2.0$. After the reinitialization of matrix W, it might have the following values:

2.0	0.0	2.0
3.0	0.0	0.0

So, finally, the child of matrix V after mutation is

0.0	0.0	5.0	0.0	3.0
0.0	**2.0**	**0.0**	0.0	**2.0**
0.0	0.0	5.0	7.0	0.0
3.0	**3.0**	**0.0**	0.0	**0.0**

Mutation-2. This version is identical to mutation-1 except that in recalculating the contents of the chosen sub-matrix a modified version of the *initialization* routine is used.

It is changed from that described earlier in the following way: the line 6

set val $\leftarrow$ min(sour[i], dest[j])

is replaced by

set $\text{val}_1 \leftarrow$ min(sour[i], dest[j])
if (i is the last available row) **or** (j is the last available column)
then
val $\leftarrow \text{val}_1$
else
set val $\leftarrow$ random (real) number from $\langle 0, \text{val}_1 \rangle$
fi

This change provides real numbers instead of integers and zeros but the procedure must be further modified as it currently produces a matrix which may violate the constraints.

For example, using the matrix from the first example, suppose that the sequence of selected numbers is $\langle 3, 6, 12, 8, 10, 1, 2, 4, 9, 11, 7, 5 \rangle$ and that the first real number generated for number 3 (first row, third column) is 7.3 (which is within the range $\langle 0.0, \text{min}(\text{sour}[1], \text{dest}[3]) \rangle = \langle 0.0, 15.0 \rangle$). The second random real number for 6 (second row, second column) is 12.1, and the rest of the real numbers generated by the new initialization algorithm are 3.3, 5.0, 1.0, 3.0, 1.9, 1.7, 0.4, 0.3, 7.4, 0.5. The resulting matrix is

	5.0	**15.0**	**15.0**	**10.0**
15.0	3.0	1.9	7.3	1.7
25.0	0.5	12.1	7.4	5.0
5.0	0.4	1.0	0.3	3.3

Only by adding 1.1 to the element x_{11} can we satisfy the constraints. So we need to add an additional (last) line to the mutation-2 algorithm:

make necessary additions

This completes the modification of the *initialization* procedure.

G9.8.3 Arithmetical crossover

Starting with two parents (matrices U and V) the arithmetical crossover operator will produce two children X and Y, where $X = c_1U + c_2V$ and $Y = c_1V + c_2U$ (where $c_1, c_2 \geq 0$ and $c_1 + c_2 = 1$). As the constraint set is convex this operation ensures that both of the children are feasible if both parents are. This is a significant simplification of the linear case where there was an additional requirement to maintain all components of the matrix as integers.

G9.8.4 Testing the algorithm

It is clear that all operators of Genetic-2n maintain feasibility of potential solutions: arithmetical crossover produces a point between two feasible points of the convex search space and both mutations are restricted to submatrices only to ensure no change in marginal sums.

In addition to the set of control parameters used for the linear case (such as population size, mutation and crossover rates, random number starting seed) we also need the crossover proportions, c_1 and c_2, and m_1, a parameter to determine the proportion of mutation-1 in the mutations applied.

In testing the Genetic-2n algorithm on the linear transportation problem (see Vignaux and Michalewicz 1991) we could have compared its solution with the known optimum found using the standard algorithm. That is, we could have determined how efficient the GA was in absolute terms. Once we move to nonlinear objective functions, the optimum may not be known. Testing is reduced to comparing the results with those of other nonlinear solution methods that may themselves have converged to a local optimum.

We have chosen to compare the Genetic-2n algorithm method with the GAMS (General Algebraic Modeling System), a package for the construction and solution of mathematical programming models (Brooke *et al* 1988), with MINOS optimizer. GAMS represents a typical example of an industry-standard efficient method of solution. This system, being essentially a gradient-controled method, found some of the problems we set up difficult or impossible to solve.

The behavior of nonlinear optimization algorithms depends markedly on the form of the objective function. It is clear that different solution techniques may respond quite differently. For purposes of testing, we have arbitrarily classified potential objective functions into those that might conceivably be seen in practical OR problems (practical), those that are mainly seen in textbooks on optimization (reasonable) and those that are more often seen as difficult test cases for optimization techniques (other).

Practical functions:

- function A

$$A(x) = \begin{cases} 0 & \text{if } 0 < x \leq S \\ c_{ij} & \text{if } S < x \leq 2S \\ 2c_{ij} & \text{if } 2S < x \leq 3S \\ 3c_{ij} & \text{if } 3S < x \leq 4S \\ 4c_{ij} & \text{if } 4S < x \leq 5S \\ 5c_{ij} & \text{if } 5S < x \end{cases}$$

  where S is less than a typical nonzero x value.
- function B

$$B(x) = \begin{cases} c_{ij}\frac{x}{S} & \text{if } 0 \leq x \leq S \\ c_{ij} & \text{if } S < x \leq 2S \\ c_{ij}\left(1 + \frac{x-2S}{S}\right) & \text{if } 2S < x \end{cases}$$

- where S is of the order of a typical nonzero x value.

Reasonable functions:

- function C

$$C(x) = c_{ij}x^2$$

- function D

$$D(x) = c_{ij}\sqrt{x}.$$

Other functions:

- function E

$$E(x) = c_{ij}\left(\frac{1}{1+(x-2S)^2} + \frac{1}{1+(x-\frac{9}{4}S)^2} + \frac{1}{1+(x-\frac{7}{4}S)^2}\right)$$

  where S is of the order of a typical nonzero x value.

- function F

$$F(x) = c_{ij}x\left(\sin\left(x\frac{5\pi}{4S}\right)+1\right)$$

  where S is of the order of a typical nonzero x value.

The objective for the transportation problem was then of the form

$$\sum_{ij} f(x_{ij})$$

where $f(x)$ is one of the functions above and the c_{ij} parameters are obtained from the parameter matrix and S from the attributes of the problem to be tested. S is approximated from the average nonzero arc flow determined from a number of preliminary runs to make sure the flows occurred in the interesting part of the objective function.

For the main set of experiments, five 10×10 transportation matrices were used with each function. They were constructed from a set of independent uniformly distributed c_{ij} values and randomly chosen source and destination vectors with a total flow of 100 units. Each function–matrix combination was given five runs using different random-number starting seeds for the GA. Problems were run for 10 000 generations. For function A, S was set to two, while for functions B, E, and F a value of five was used.

The 10×10 node problems reach the limit of the student version of GAMS (where allowable problem size is restricted). From a listing of some example problems tested on the GAMS/MINOS system given by Brooke *et al* (1988), it appears that with the full version (where problem size is limited by available memory and internal limits) on a 640 kbyte memory AT computer, a 25×25 node problem should be possible. Note that an $N \times N$ node problem would be formulated by GAMS/MINOS as having N^2 variables, $2N$ constraints and a nonlinear objective function. Clearly, larger problems could be formulated on bigger systems (especially a mainframe) or with specialized solvers.

However using much larger problems to compare the genetic system with nonlinear programming type solvers may be of limited value. Results of the 10×10 runs demonstrate the tendency for GAMS/MINOS (and presumably, similar systems) to fall into local (nonglobal) optima. Ignoring the time spent evaluating the objective function and using the number of solutions tested as the measure of time it is clear that standard nonlinear programming techniques will always 'finish' faster than genetic systems. This is because they typically explore only a particular path within the current local optimum zone. They will do well only if the local optimum is a relatively good one.

A set of parameters were chosen for Genetic-2n after experience with the linear problems and on the basis of tuning runs with the nonlinear problems. The population size was fixed at 40. The mutation rate was $p_m = 20\%$ with the proportion of mutation-1 being 50%, and the crossover rate was $p_c = 5\%$. The constants used for crossover were $c_1 = 0.35$ and $c_2 = 0.65$.

It may appear that the chosen mutation rate is too high and the crossover rate too low in comparison with classical GAs. However, our operators are different from the classical ones, because (i) we select parents for mutations and crossovers, that is, the *whole* structure (as opposed to single bits) undergoes mutation, and (ii) mutation-1 creates an offspring 'pushing' the parent towards the surface of the solution space, whereas crossover and mutation-2 'push' the offspring towards the center of the solution space.

The Genetic-2n system was run on Sun SPARCstation 1 computers while GAMS was run on an Olivetti 386. Although speed comparisons between the two machines are difficult it should be noted that in general GAMS finished each run well before the genetic system. An exception is function A (in which GAMS evaluates numerous arctangent functions) where the genetic algorithm took no more than 15 minutes to complete while GAMS averaged about twice that. For functions A, B, and D, where the extra GAMS modification parameter meant that multiple runs had to be performed to find its best solution, the genetic system overall was much faster.

A typical comparison of the optima between Genetic-2n (averaged over five seeds) and GAMS is shown below for a *single* problem.

Function	GAMS	Genetic-2n	% difference
A	281.0	202.0	−28.1
B	180.8	163.0	−9.8
C	4402.0	4556.2	+3.5
D	408.4	391.1	−4.2
E	145.1	79.2	−45.4
F	1200.8	201.9	−83.2

For the class of 'practical' problems, *A* and *B*, Genetic-2n is, on average, better than GAMS by 24.5% for *A* and by 11.5% for *B*. For the 'reasonable' functions the results were different. For *C* (the square function), the genetic system performed worse by 7.5% while for *D* (the square-root function), the genetic system was better by just 2.0%, on average. For the 'other' functions, *E* and *F*, the genetic system dominates: it resulted in improvements of 33.0 and 54.5% over GAMS, averaging over the *five* problems.

Genetic-2n was specifically tailored to transportation problems but an important characteristic is that it handles any type of cost function (which need not even be continuous). It is also possible to modify it to handle many similar operations research problems including allocation and some scheduling problems. This seems to be a promising research direction which may result in a generic technique for solving matrix-based constrained optimization problems.

References

Brooke A, Kendrick D and Meeraus A 1988 *GAMS: A User's Guide* (Belmont, CA: Scientific Press)

Fogel L J, Owens A J and Walsh M J 1966 *Artificial Intelligence Through Simulated Evolution* (Chichester: Wiley)

Michalewicz Z 1996 *Genetic Algorithms + Data Structures = Evolution Programs* 3rd edn (New York: Springer)

Michalewicz Z, Vignaux G A and Hobbs M 1991 A non-standard genetic algorithm for the nonlinear transportation problem *ORSA J. Comput.* **3** 307–16

Taha H A 1987 *Operations Research: an Introduction* 4th edn (London: Collier Macmillan)

Vignaux G A and Michalewicz Z 1991 A genetic algorithm for the linear transportation problem *IEEE Trans. Syst. Man Cybernet.* **SMC-21** 445–52

G9.9 Numerical optimization: handling nonlinear constraints

Zbigniew Michalewicz

Abstract

This case study surveys several techniques that have emerged in the evolutionary computation community to handle numerical optimization problems with nonlinear constraints.

G9.9.1 Introduction

Evolutionary computation techniques have received much attention regarding their potential as optimization techniques for complex numerical functions. However, they did not make a significant breakthrough in the area of nonlinear programming due to the fact that they did not address the issue of constraints in a systematic way. Only recently several methods have been proposed for handling nonlinear constraints by evolutionary algorithms for numerical optimization problems; however, these methods had several drawbacks and the experimental results were, on many test cases, disappointing.

This section surveys several approaches which have emerged recently in the evolutionary computation community and describes the most recent nonlinear programming tool: Genocop III (Michalewicz and Nazhiyath 1995), which is based on concepts of coevolution and repair algorithms.

The general nonlinear programming (NLP) problem is to find $\boldsymbol{x}$ so as to

$$\text{optimize } f(\boldsymbol{x}), \boldsymbol{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$$

where $\boldsymbol{x} \in \mathcal{F} \subseteq \mathcal{S}$. The set $\mathcal{S} \subseteq \mathbb{R}^n$ defines the search space and the set $\mathcal{F} \subseteq \mathcal{S}$ defines a *feasible* search space. Usually, the search space $\mathcal{S}$ is defined as a n-dimensional rectangle in $\mathbb{R}^n$ (domains of variables defined by their lower and upper bounds):

$$l(i) \leq x_i \leq u(i) \qquad 1 \leq i \leq n$$

whereas the feasible set $\mathcal{F} \subseteq \mathcal{S}$ is defined by a set of additional $m \geq 0$ constraints:

$$g_j(\boldsymbol{x}) \leq 0 \qquad \text{for } j = 1, \ldots, q$$

and

$$h_j(\boldsymbol{x}) = 0 \qquad \text{for } j = q+1, \ldots, m.$$

It is convenient to divide all constraints into four subsets: linear equations LE, linear inequalities LI, nonlinear equations NE, and nonlinear inequalities NI. Of course, $g_j \in \text{LI} \cup \text{NI}$ and $h_j \in \text{LE} \cup \text{NE}$.

The NLP problem, in general, is intractable. If the objective function f and functions g_j and h_j expressing constraints are arbitrary, then there is little choice apart from methods based on exhaustive search. This is the reason for identifying several special cases of the NLP. One of the cases studied is the one where all functions g_j and h_j are linear; such problems are called *linearly constrained optimization* problems. If, additionally, the objective function f is at most quadratic, the problem is called the *quadratic programming* problem. The best known special case of quadratic programming is where the objective

function f is linear as well; this problem is called the *linear programming* problem. There is also an important special case, called *unconstrained optimization*, where there are no constraints at all, that is, $m = 0$ and $\mathcal{F} = \mathcal{S}$.

Evolutionary algorithms are global methods, which aim at complex objective functions (e.g. nondifferentiable or discontinuous). However, most research on applications of evolutionary computation techniques to nonlinear programming problems has been concerned with complex objective functions with $\mathcal{F} = \mathcal{S}$. Several test functions used by various researchers during the last 20 years considered only domains of n variables; this was the case with five test functions F1–F5 proposed by De Jong (1975), as well as with many other test cases proposed since then (Eshelman and Schaffer 1993, Fogel and Stayton 1994, Wright 1991).

Only recently several approaches emerged which aimed at general nonlinear programming problems. Most of them are based on the concept of *penalty functions*, which penalize infeasible solutions, that is, C5.2

$$\text{eval}(\boldsymbol{x}) = \begin{cases} f(\boldsymbol{x}) & \text{if } \boldsymbol{x} \in \mathcal{F} \\ f(\boldsymbol{x}) + \text{penalty}(\boldsymbol{x}) & \text{otherwise} \end{cases}$$

where penalty($\boldsymbol{x}$) is zero if no violation occurs, and is positive, otherwise. In most methods, a set of functions f_j ($1 \leq j \leq m$) is used to construct the penalty; the function f_j measures the violation of the jth constraint in the following way:

$$f_j(\boldsymbol{x}) = \begin{cases} \max\{0, g_j(\boldsymbol{x})\} & \text{if } 1 \leq j \leq q \\ |h_j(\boldsymbol{x})| & \text{if } q+1 \leq j \leq m. \end{cases}$$

(In the rest of the paper we assume minimization problems.) However, these methods differ in many important details of how the penalty function is designed and applied to infeasible solutions. In the following paragraphs we discuss these methods in turn.

G9.9.2 The Genocop system

The Genocop (for GEnetic algorithm for Numerical Optimization of COnstrained Problems) system was developed (Michalewicz 1996; see also Section G9.1) for problems with linear constraints; the system gave G9.1 surprisingly good performance on many test functions. The method can be generalized to handle nonlinear constraints provided that the resulting feasible search space $\mathcal{F}$ is convex. However, the weakness of the method lies in its inability to deal with nonconvex search spaces (i.e. to deal with nonlinear constraints in general).

The method proposed by Homaifar *et al* (1994) assumes that for every constraint we establish a family of intervals which determine an appropriate penalty coefficient. It works as follows:

- for each constraint, create several (ℓ) levels of violation
- for each level of violation and for each constraint create a penalty coefficient R_{ij} ($i = 1, 2, \ldots, \ell$, $j = 1, 2, \ldots, m$); higher levels of violation require larger values of this coefficient
- start with a random population of individuals (feasible or infeasible)
- evolve the population; evaluate individuals using the following formula

$$\text{eval}(\boldsymbol{x}) = f(\boldsymbol{x}) + \sum_{j=1}^{m} R_{ij} f_j^2(\boldsymbol{x}).$$

The weakness of the method is in the number of parameters: for m constraints the method requires m parameters to establish the number of intervals for each constraint (in Homaifar *et al* 1994, these parameters are the same for all constraints equal to $\ell = 4$), ℓ parameters for each constraint (i.e. ℓm parameters in total) that represent boundaries of intervals (levels of violation), and ℓ parameters for each constraint (ℓm parameters in total) that represent the penalty coefficients R_{ij}, so the method requires $m(2\ell+1)$ parameters in total to handle m constraints. In particular, for $m = 5$ constraints and $\ell = 4$ levels of violation, we need to set 45 parameters! Clearly, the results are parameter dependent. It is quite likely that for a given problem there exists an optimal set of parameters for which the system returns feasible near-optimum solution; however, it might be quite hard to find it.

The method proposed by Joines and Houck (1994) assumes dynamic penalties. Individuals are evaluated (at the iteration t) by the following formula:

$$\text{eval}(\boldsymbol{x}) = f(\boldsymbol{x}) + (Ct)^{\alpha} \sum_{j=1}^{m} f_j^{\beta}(\boldsymbol{x})$$

where C, α, and β are constants. A reasonable choice for these parameters is $C = 0.5$, $\alpha = \beta = 2$. The method requires a much smaller number (independent of the number of constraints) of parameters than the first method. Also, instead of defining several levels of violation, the pressure on infeasible solutions is increased due to the $(Ct)^{\alpha}$ component of the penalty term: towards the end of the process (for high values of the generation number t), this component assumes large values.

The method proposed by Schoenauer and Xanthakis (1993) was based on a 'behavioral memory' approach. It works as follows:

- Start with a random population of individuals (feasible or infeasible).
- Set $j = 1$ (j is a constraint counter).
- Evolve this population with $\text{eval}(\boldsymbol{x}) = f_j(\boldsymbol{x})$, until a given percentage of the population (the so-called flip threshold ϕ) is feasible for this constraint.
- Set $j = j + 1$.
- The current population is the starting point for the next phase of the evolution, where $\text{eval}(\boldsymbol{x}) = f_j(\boldsymbol{x})$. During this phase, points that do not satisfy one of the first, second, ..., or $(j-1)$th constraint are eliminated from the population. The stop criterion is again the satisfaction of the jth constraint by the flip threshold percentage ϕ of the population.
- If $j < m$, repeat the last two steps, otherwise ($j = m$) optimize the objective function, that is, $\text{eval}(\boldsymbol{x}) = f(\boldsymbol{x})$, rejecting infeasible individuals.

The method requires a linear order of all constraints which are processed in turn. The influence of the order of constraints on the results of the algorithm is unclear; some experiments (Michalewicz 1995) indicated that different orders provide different results (different in the sense of the total running time and precision).

In total, the method requires three parameters: the sharing factor σ, the flip threshold ϕ, and a particular order of constraints. The method is very different to the previous two methods, and, in general, is different from other penalty approaches, since it considers only one constraint at the time. Also, in the last step of the algorithm the method optimizes the objective function f itself without any penalty component.

The method (Genocop II) described by Michalewicz and Attia (1994) works as follows:

- Divide all constraints into four subsets: LE, LI, NE, and NI.
- Select a random single point as a starting point (the initial population consists of copies of this single individual). This initial point satisfies linear constraints (LE and LI).
- Set the initial temperature $\tau = \tau_0$.
- Evolve the population using the following formula:

$$\text{eval}(\boldsymbol{x}, \tau) = f(\boldsymbol{x}) + \frac{1}{2\tau} \sum_{j=1}^{m} f_j^2(\boldsymbol{x}).$$

- If $\tau < \tau_f$, stop, otherwise:
  - decrease temperature τ
  - the best solution serves as a starting point of the next iteration
  - repeat the previous step of the algorithm.

This is the only method described here which distinguishes between linear and nonlinear constraints. The algorithm maintains the feasibility of all linear constraints using a set of closed operators, which converts a feasible solution (feasible in terms of linear constraints only) into another feasible solution. At every iteration the algorithm considers active constraints only; the pressure on infeasible solutions is increased due to the decreasing values of temperature τ.

The method has an additional unique feature: it starts from a single point. (This feature, however, is not essential. The only important requirement is that the next population contains the best individual from the previous population.) Consequently, it is relatively easy to compare this method with other classical optimization methods whose performances are tested (for a given problem) from some starting point.

The method requires starting and 'freezing' temperatures, τ_0 and τ_f, respectively, and the cooling scheme to decrease temperature τ. Standard values are $\tau_0 = 1$, $\tau_{i+1} = 0.1\tau_i$, with $\tau_f = 0.000\,001$.

The method developed by Powell and Skolnick (1993) is a classical penalty method with one notable exception. Each individual is evaluated by the formula

$$\text{eval}(\boldsymbol{x}) = f(\boldsymbol{x}) + r\sum_{j=1}^{m} f_j(\boldsymbol{x}) + \theta(t, \boldsymbol{x})$$

where r is a constant; however, there is also a component $\theta(t, \boldsymbol{x})$. This is an additional iteration-dependent function which influences the evaluations of infeasible solutions. The point is that the method distinguishes between feasible and infeasible individuals by adopting an additional heuristic rule (suggested earlier by Richardson *et al* 1989): for any feasible individual $\boldsymbol{x}$ and any infeasible individual $\boldsymbol{y}$: $\text{eval}(\boldsymbol{x}) < \text{eval}(\boldsymbol{y})$, that is, any feasible solution is better than any infeasible one. This can be achieved in many ways; one possibility is to set

$$\theta(t, \boldsymbol{x}) = \begin{cases} 0 & \text{if } \boldsymbol{x} \in \mathcal{F} \\ \max\left\{0, \max\limits_{\boldsymbol{x}\in\mathcal{F}}\{f(\boldsymbol{x})\} - \min\limits_{\boldsymbol{x}\in\mathcal{S}-\mathcal{F}}\left\{f(\boldsymbol{x}) + r\sum\limits_{j=1}^{m} f_j(\boldsymbol{x})\right\}\right\} & \text{otherwise.} \end{cases}$$

In other words, infeasible individuals have increased penalties: their values cannot be better than the value of the worst feasible individual (i.e. $\max_{\boldsymbol{x}\in\mathcal{F}}\{f(\boldsymbol{x})\}$). Powell and Skolnick achieved the same result by mapping evaluations of feasible solutions into the interval $(-\infty, 1)$ and infeasible solutions into the interval $(1, \infty)$. For ranking and tournament selections this implementational difference is not important.

The method developed by Bean and Hadj-Alouane (1992), like the previous method, uses a penalty function; however, one component of the penalty function takes a feedback from the search process. Each individual is evaluated by the formula

$$\text{eval}(\boldsymbol{x}) = f(\boldsymbol{x}) + \lambda(t)\sum_{j=1}^{m} f_j^2(\boldsymbol{x})$$

where $\lambda(t)$ is updated every generation t in the following way:

$$\lambda(t+1) = \begin{cases} (1/\beta_1)\cdot\lambda(t) & \text{if } \boldsymbol{b}(i) \in \mathcal{F} \text{ for all } t-k+1 \le i \le t \\ \beta_2\cdot\lambda(t) & \text{if } \boldsymbol{b}(i) \in \mathcal{S}-\mathcal{F} \text{ for all } t-k+1 \le i \le t \\ \lambda(t) & \text{otherwise} \end{cases}$$

where $\boldsymbol{b}(i)$ denotes the best individual, in terms of function *eval*, in generation i, $\beta_1, \beta_2 > 1$ and $\beta_1 \neq \beta_2$ (to avoid cycling). In other words, the method (i) decreases the penalty component $\lambda(t+1)$ for the generation $t+1$, if all best individuals in the last k generations were feasible, and (ii) increases penalties, if all best individuals in the last k generations were infeasible. If there are some feasible and infeasible individuals as best individuals in the last k generations, $\lambda(t+1)$ remains without change.

The intuitive reason behind adaptation of penalties in the method of Bean and Hadj-Alouane is similar to the reason behind the one-fifth success rule of evolution strategies: the increased efficiency of the search. In evolution strategies, if successful, the search would continue in 'larger' steps; if not, the steps would be shorter. In the method of Bean and Hadj-Alouane, if constraints do not pose a problem, the search will continue with decreased penalties, if not, the penalties will be increased. The presence of both feasible and infeasible individuals in the set of best individuals in the last k generations means that the current value of penalty component $\lambda(t)$ is set correctly.

We can consider also a death penalty method, which rejects infeasible individuals; the method has been used by evolution strategies and simulated annealing.

Several other constraint handling methods deserve also some attention. For example, some methods make use of the values of objective function f and penalties f_j as elements of a vector and apply multi-objective techniques to minimize all components of the vector (Surry *et al* 1995). However, analysis of Schaffer's VEGA system indicated the equivalence between multiobjective optimization and linear combination of f and f_j (Richardson *et al* 1989). Also, an interesting approach was recently reported by Paredis (1994). The method (described in the context of constraint satisfaction

problems) is based on a *coevolutionary model*, where a population of potential solutions coevolves with a population of constraints: fitter solutions satisfy more constraints, whereas fitter constraints are violated by more solutions. There is some development connected with generalizing the concept of 'ant colonies' (Colorni *et al* 1991), which were originally proposed for order-based problems, to numerical domains (Bilchev and Parmee 1995); first experiments on various test problems gave very good results (Bilchev 1995). Smith and Tate (1993) experimented with dynamic penalties, where the penalty measure depends on the number of violated constraints, the best feasible objective function found, and the best objective function value found. Le Riche *et al* (1995) proposed a segregated genetic algorithm which uses a double-penalty strategy. Two populations are evolved: one with small, the other with large penalties. These two populations are merged every generation; the best individuals survive and reproduce. In this way the algorithm is less sensitive to the choice of the penalty parameters. It is also possible to incorporate the knowledge of the constraints of the problem into the belief space of cultural algorithms (Reynolds 1994); such algorithms provide a possibility of conducting an efficient search of the feasible search space (Reynolds *et al* 1995). However, all these methods are in early stages of their development, or they have not been applied to the general NLP, so we do not discuss them any further.

C5.6.3

G9.9.3 Genocop III

Recently (Michalewicz and Nazhiyath 1995) a new system, Genocop III, was developed. Genocop III handles all kinds of constraint: LE, LI, NE, and NI. As in the original *Genocop*, linear constraints are eliminated, the number of variables is reduced, and LI are modified accordingly. All points included in the initial population satisfy linear constraints; specialized operators maintain their feasibility (in the sense of linear constraints) from one generation to the next. We denote a set of points which satisfy linear constraints by $\mathcal{F}_l \subseteq \mathcal{S}$.

G9.1

NE require an additional parameter (γ) to define the precision of the system. All NE $h_j(\boldsymbol{x}) = 0$ (for $j = q+1, \ldots, m$) are replaced by a pair of inequalities:

$$-\gamma \leq h_j(\boldsymbol{x}) \leq \gamma$$

so we deal only with NI. These NI further restrict the set $\mathcal{F}_l$: they define the fully feasible part $\mathcal{F} \subseteq \mathcal{F}_l$ of the search space $\mathcal{S}$.

Genocop III incorporates the original Genocop system, but also extends it by maintaining two separate populations, where a development in one population influences evaluations of individuals in the other population. The first population P_s consists of so-called search points from $\mathcal{F}_l$ which satisfy linear constraints of the problem. As mentioned earlier, the feasibility (in the sense of linear constraints) of these points is maintained by specialized operators (see Section G9.1). The second population P_r consists of so-called reference points from $\mathcal{F}$; these points are fully feasible, that is, they satisfy *all* constraints. If Genocop III has difficulties in locating such a reference point for the purpose of initialization, the user is prompted for it. In cases, where the ratio $|\mathcal{F}|/|\mathcal{S}|$ is very small, it may happen that the initial set of reference points consists of a multiple copies of a single feasible point.

Reference points $\boldsymbol{r}$ from P_r, being feasible, are evaluated directly by the objective function (i.e. $\text{eval}(\boldsymbol{r}) = f(\boldsymbol{r})$). On the other hand, search points from P_s are 'repaired' for evaluation and the repair process works as follows. Assume, there is a search point $\boldsymbol{s} \in P_s$. If $\boldsymbol{s} \in \mathcal{F}$, then $\text{eval}(\boldsymbol{s}) = f(\boldsymbol{s})$, since $\boldsymbol{s}$ is fully feasible. Otherwise (i.e. $\boldsymbol{s} \notin \mathcal{F}$), the system selects one of the reference points, say $\boldsymbol{r}$ from P_r and creates a sequence of points $\boldsymbol{z}$ from a segment between $\boldsymbol{s}$ and $\boldsymbol{r}$: $\boldsymbol{z} = a\boldsymbol{s} + (1-a)\boldsymbol{r}$. This can be done either (i) in a random way by generating random numbers a from the range $(0, 1)$, or (ii) in a deterministic way by setting $a_i = 1/2, 1/4, 1/8, \ldots$ until a feasible point is found. Note, that all such generated points $\boldsymbol{z} \in \mathcal{F}_l$. Once a fully feasible $\boldsymbol{z}$ is found, $\text{eval}(\boldsymbol{s}) = \text{eval}(\boldsymbol{z}) = f(\boldsymbol{z})$. Clearly, in different generations the same search point $\mathcal{S}$ can evaluate to different values due to the random nature of the repair process.

Additionally, if $f(\boldsymbol{z})$ is better than $f(\boldsymbol{r})$, then the point $\boldsymbol{z}$ replaces $\boldsymbol{r}$ as a new reference point in the population of reference points P_r. Also, $\boldsymbol{z}$ replaces $\boldsymbol{s}$ in the population of search points P_s with some probability of replacement p_r.

The structure of Genocop III is as follows:

```
procedure Genocop III
begin
    t ← 0
    initialize P_s(t)
    initialize P_r(t)
    evaluate P_s(t)
    evaluate P_r(t)
    while (not termination-condition) do
    begin
        t ← t + 1
        select P_s(t) from P_s(t − 1)
        alter P_s(t)
        evaluate P_s(t)
        if t mod k = 0 then
        begin
            alter P_r(t)
            select P_r(t) from P_r(t − 1)
            evaluate P_r(t)
        end
    end
end
```

The procedure for evaluating (not necessarily fully feasible) search points from population P_s is as follows:

```
procedure evaluate P_s(t)
begin
    for each s ∈ P_s(t) do
        if s ∈ F
        then evaluate s (as f(s)) else
        begin
            select r ∈ P_r(t)
            generate z ∈ F
            evaluate s (as f(z))
            if f(r) > f(z) then replace r by z in P_r
            replace s by z in P_s with probability p_r
        end
end
```

Note that there is some asymmetry between the processing population of search points P_s and the population of reference points P_r: while we apply the selection procedure and operators to P_s every generation, population P_r is modified every k (parameter of the method) generations (however, some additional changes in P_r are possible during the evaluation of search points; see the evaluation procedure above. The main reason behind this arrangement is efficiency of the system: search within the feasible part of the search space $\mathcal{F}$, considered as less effective for NLP problems, is treated as a background event. Note also, that the 'selection' and 'alternation' steps are reversed in the evolution loop for the P_r: due to a low probability of generating feasible offspring, first parent individuals reproduce and later the best feasible individuals (from both parents and offspring) are selected for the next population (i.e. $\mu + \lambda$ selection from evolution strategies).

Genocop III avoids many disadvantages of other systems. It uses the objective function for evaluation of fully feasible individuals only, so the evaluation function is not distorted as in methods based on penalty functions. It always returns a feasible solution. A feasible search space $\mathcal{F}$ is searched (population P_r) by making references from the search points and by application of operators (every some number of generations, k, in the evaluation procedure. The neighborhoods of better reference points are explored

more often. Some fully feasible points are moved into the population of search points P_s (replacement process), where they undergo additional transformation by specialized operators.

On the other hand, there are a few additional parameters: the population sizes of search and reference points, probability of replacement p_r, frequency k of application of operators to the population of reference points, the method of repair (random versus deterministic), and the selection method of reference points for the repair process.

G9.9.4 Test cases

In order to evaluate the method, a set of test problems has been carefully selected to illustrate the performance of the algorithm and to indicate its degree of success. The eight test cases include quadratic, nonlinear, and discontinuous functions with several linear constraints. We used the following parameters for all experiments: population sizes for both populations (search points and reference points) were set to 70, there were 28 parents selected in each generation, $b = 6$ (coefficient for non-uniform mutation), the repair method was random, with probability of replacement $p_r = 0.15$, and the parameter k was set to infinity (i.e. there were no operators applied directly to the population of reference points; all improvements came from the repair process). Genocop III was executed ten times for each test case.

Test case G9.9.1. The problem (Floudas and Pardalos 1987) is to minimize a function

$$f(\boldsymbol{x}) = 5x_1 + 5x_2 + 5x_3 + 5x_4 - 5\sum_{i=1}^{4} x_i^2 - \sum_{i=5}^{13} x_i$$

subject to

$$\begin{array}{lll}
2x_1 + 2x_2 + x_{10} + x_{11} \leq 10 & 2x_1 + 2x_3 + x_{10} + x_{12} \leq 10 & 2x_2 + 2x_3 + x_{11} + x_{12} \leq 10 \\
-8x_1 + x_{10} \leq 0 & -8x_2 + x_{11} \leq 0 & -8x_3 + x_{12} \leq 0 \\
-2x_4 - x_5 + x_{10} \leq 0 & -2x_6 - x_7 + x_{11} \leq 0 & -2x_8 - x_9 + x_{12} \leq 0 \\
0 \leq x_i \leq 1 \quad i = 1, \ldots, 9 & 0 \leq x_i \leq 100 \quad i = 10, 11, 12 & 0 \leq x_{13} \leq 1.
\end{array}$$

The problem has nine linear constraints; the function f is quadratic with its global minimum at

$$\boldsymbol{x}^* = (1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 3, 3, 1)$$

where $f(\boldsymbol{x}^*) = -15$. Six (out of nine) constraints are active at the global optimum (all except the following three: $-8x_1 + x_{10} \leq 0$, $-8x_2 + x_{11} \leq 0$, $-8x_3 + x_{12} \leq 0$). Since Genocop III is based on the original Genocop, which handles linear constraints by feasibility-preserving operators, it found the optimum in all runs.

Test case G9.9.2. The problem (Hock and Schittkowski 1981) is to minimize a function

$$f(\boldsymbol{x}) = x_1 + x_2 + x_3$$

where

$$\begin{array}{ll}
1 - 0.0025(x_4 + x_6) \geq 0 & x_1x_6 - 833.332\,52x_4 - 100x_1 + 83\,333.333 \geq 0 \\
1 - 0.0025(x_5 + x_7 - x_4) \geq 0 & x_2x_7 - 1250x_5 - x_2x_4 + 1250x_4 \geq 0 \\
1 - 0.01(x_8 - x_5) \geq 0 & x_3x_8 - 1\,250\,000 - x_3x_5 + 2500x_5 \geq 0 \\
100 \leq x_1 \leq 10\,000 & 1000 \leq x_i \leq 10\,000 \quad i = 2, 3 \quad 10 \leq x_i \leq 1000 \quad i = 4, \ldots, 8.
\end{array}$$

The problem has three linear and three nonlinear constraints; the function f is linear and has its global minimum at

$$\boldsymbol{x}^* = (579.3167, 1359.943, 5110.071, 182.0174, 295.5985, 217.9799, 286.4162, 395.5979)$$

where $f(\boldsymbol{x}^*) = 7\,049.330\,923$. All six constraints are active at the global optimum. Genocop III (in 10 000 generations) found the following solution:

$$\boldsymbol{x} = (122.771, 1280.199, 5888.056, 125.6815, 264.4778, 274.318, 261.2037, 364.4778)$$

where $f(\boldsymbol{x}) = 7\,291.025\,390\,62$.

It is interesting to observe a strong correlation between the performance of the system and the number of linear constraints (note that some—or all—linear constraints can be defined in the system as nonlinear ones): the results were the best when all constraints were defined as nonlinear! It seems that linear constraints of this problem prevented the system from moving closer to the optimum (all search points must satisfy linear and domain constraints). This is an interesting example of the damaging effect of limiting the population to the feasible (with respect to linear constraints) region only.

Test case G9.9.3. The problem (Hock and Schittkowski 1981) is to minimize a function

$$f(\boldsymbol{x}) = (x_1 - 10)^2 + 5(x_2 - 12)^2 + x_3^4 + 3(x_4 - 11)^2 + 10x_5^6 + 7x_6^2 + x_7^4 - 4x_6x_7 - 10x_6 - 8x_7$$

where

$$127 - 2x_1^2 - 3x_2^4 - x_3 - 4x_4^2 - 5x_5 \geq 0 \qquad 282 - 7x_1 - 3x_2 - 10x_3^2 - x_4 + x_5 \geq 0$$
$$196 - 23x_1 - x_2^2 - 6x_6^2 + 8x_7 \geq 0 \qquad -4x_1^2 - x_2^2 + 3x_1x_2 - 2x_3^2 - 5x_6 + 11x_7 \geq 0$$
$$-10.0 \leq x_i \leq 10.0 \quad i = 1, \ldots, 7.$$

The problem has four nonlinear constraints; the function f is nonlinear and has its global minimum at

$$\boldsymbol{x}^* = (2.330\,499, 1.951\,372, -0.477\,541\,4, 4.365\,726, -0.624\,487\,0, 1.038\,131, 1.594\,227)$$

where $f(\boldsymbol{x}^*) = 680.630\,057\,3$. Two (out of four) constraints are active at the global optimum (the first and the last one). Genocop III (in 10 000 generations) gave a very good performance; the best solution found was

$$\boldsymbol{x} = (2.342\,349, 1.958\,588, -0.415\,317, 4.338\,925, -0.601\,853, 1.035\,198, 1.594\,577)$$

where $f(\boldsymbol{x}^*) = 680.659\,301\,7$, whereas the worst solution (out of ten runs) gave the value of 680.796 875 (which is better than the best value found by several penalty-based methods reported by Michalewicz 1995).

Test case G9.9.4. The problem (Hock and Schittkowski 1981) is to minimize a function

$$f(\boldsymbol{x}) = x_1^2 + x_2^2 + x_1x_2 - 14x_1 - 16x_2 + (x_3 - 10)^2 + 4(x_4 - 5)^2 + (x_5 - 3)^2 + 2(x_6 - 1)^2 + 5x_7^2 + 7(x_8 - 11)^2 + 2(x_9 - 10)^2 + (x_{10} - 7)^2 + 45$$

where

$$105 - 4x_1 - 5x_2 + 3x_7 - 9x_8 \geq 0 \qquad -3(x_1 - 2)^2 - 4(x_2 - 3)^2 - 2x_3^2 + 7x_4 + 120 \geq 0$$
$$-10x_1 + 8x_2 + 17x_7 - 2x_8 \geq 0 \qquad -x_1^2 - 2(x_2 - 2)^2 + 2x_1x_2 - 14x_5 + 6x_6 \geq 0$$
$$8x_1 - 2x_2 - 5x_9 + 2x_{10} + 12 \geq 0 \qquad -5x_1^2 - 8x_2 - (x_3 - 6)^2 + 2x_4 + 40 \geq 0$$
$$3x_1 - 6x_2 - 12(x_9 - 8)^2 + 7x_{10} \geq 0 \qquad -0.5(x_1 - 8)^2 - 2(x_2 - 4)^2 - 3x_5^2 + x_6 + 30 \geq 0$$
$$-10.0 \leq x_i \leq 10.0 \quad i = 1, \ldots, 10.$$

The problem has three linear and five nonlinear constraints; the function f is quadratic and has its global minimum at

$$\boldsymbol{x}^* = (2.171\,00, 2.363\,68, 8.773\,93, 5.095\,98, 0.990\,655, 1.430\,57, 1.321\,64, 9.828\,73, 8.280\,09, 8.375\,93)$$

where $f(\boldsymbol{x}^*) = 24.306\,209\,1$. Six (out of eight) constraints are active at the global optimum (all except the last two). Genocop III (in 5000 generations) gave a reasonable performance; the best solution found was

$$\boldsymbol{x} = (2.251\,32, 2.468\,48, 8.293\,99, 5.176\,34, 1.146\,61, 1.740\,87, 1.307\,43, 9.730\,50, 8.338\,66, 8.309\,83)$$

where $f(\boldsymbol{x}) = 25.336\,544\,04$.

Test case G9.9.5. The problem (Keane 1994) is to maximize a function

$$f(\boldsymbol{x}) = \left| \frac{\sum_{i=1}^{n} \cos^4(x_i) - 2\prod_{i=1}^{n} \cos^2(x_i)}{\left(\sum_{i=1}^{n} \mathrm{i}x_i^2\right)^{1/2}} \right|$$

where

$$\prod_{i=1}^{n} x_i > 0.75 \qquad \sum_{i=1}^{n} x_i < 7.5n \qquad 0 < x_i < 10 \quad \text{for } 1 \le i \le n.$$

The problem has two nonlinear constraints; the function f is nonlinear and its global maximum is unknown. Genocop III was run for cases of $n = 20$ and $n = 50$. In the former case, the best solution found (in 10 000 generations) was

$$\begin{aligned}\boldsymbol{x} = (&3.163\,113\,59, 3.131\,504\,30, 3.095\,158\,58, 3.060\,165\,88, 3.031\,035\,66,\\ &2.991\,585\,49, 2.958\,025\,93, 2.922\,858\,95, 0.486\,843\,88, 0.477\,322\,79,\\ &0.480\,444\,73, 0.487\,909\,11, 0.484\,504\,37, 0.448\,070\,32, 0.468\,777\,60,\\ &0.456\,485\,06, 0.447\,626\,08, 0.449\,139\,86, 0.443\,908\,63, 0.451\,493\,32)\end{aligned}$$

where $f(\boldsymbol{x}) = 0.803\,510\,67$. In the latter case ($n = 50$), the best solution found (in 10 000 generations) was

$$\begin{aligned}\boldsymbol{x} = (&6.280\,060\,29, 3.161\,552\,91, 3.154\,538\,15, 3.140\,851\,74, 3.128\,824\,47,\\ &3.112\,110\,85, 3.101\,705\,07, 3.087\,036\,85, 3.075\,717\,69, 3.061\,227\,32,\\ &3.050\,105\,81, 3.036\,679\,51, 3.023\,330\,45, 3.007\,210\,49, 2.994\,927\,17,\\ &2.979\,884\,62, 2.966\,370\,58, 2.955\,890\,66, 2.944\,272\,04, 2.927\,960\,40,\\ &0.409\,706\,41, 2.906\,709\,91, 0.461\,311\,19, 0.481\,933\,36, 0.467\,769\,62,\\ &0.438\,875\,50, 0.451\,810\,99, 0.446\,528\,76, 0.433\,487\,53, 0.445\,771\,43,\\ &0.423\,799\,48, 0.458\,580\,49, 0.429\,310\,50, 0.429\,286\,45, 0.429\,433\,02,\\ &0.432\,943\,61, 0.426\,633\,51, 0.434\,372\,57, 0.425\,425\,59, 0.415\,941\,54,\\ &0.432\,489\,57, 0.391\,347\,23, 0.426\,286\,88, 0.427\,743\,64, 0.418\,862\,97,\\ &0.421\,072\,63, 0.412\,153\,60, 0.418\,095\,89, 0.416\,267\,75, 0.423\,164\,07)\end{aligned}$$

where $f(\boldsymbol{x}) = 0.833\,193\,78$. (This is not the global optimum: Bilchev (1995) reported a value of the objective function of 0.8348.)

G9.9.5 Discussion

These preliminary test results are quite promising: the new system, Genocop III, performed much better than any penalty-based method (see Michalewicz 1995). Also, there are many possibilities to explore which can further enhance the performance of the system.

First of all, several experiments are required to investigate the influence of the ratio $|\mathcal{F}|/|\mathcal{S}|$ on the performance of the system. For the test cases G9.9.1–5 this ratio was 0.0111, 0.0010, 0.5121, 0.0003, and 99.9362%, respectively. (The ratio $|\mathcal{F}|/|\mathcal{S}|$ was determined experimentally by generating 1 000 000 random points from $\mathcal{S}$ and checking whether they belonged to $\mathcal{F}$.) Note also that it is possible to represent some linear constraints as nonlinear constraints; this change in the input file would change the space of reference points making it smaller and the space of linearly feasible search points would be larger (this was done for experiments with test case G9.9.2). However, it is unclear how these changes would affect the performance of the system. So far, we have not conducted any experiments with different values of parameter k; all results provided here are for a simple case where k was set to infinity (i.e. no operators were applied to the population of reference points). An additional group of experiments is required for a single parameter: probability of replacement p_r. Recently a so-called 5% rule was reported (Orvosh and Davis 1993): this heuristic rule states that in many combinatorial optimization problems an evolutionary computation technique with a repair algorithm provides the best results when 5% of repaired individuals replace their infeasible originals. In all experiments reported above, $p_r = 0.15$.

Further modification of the system would also include introduction of Boolean and integer variables, experiments with additional operators (e.g. multiparent crossovers), and experiments with adaptive frequencies of operators.

References

Bean J C and Hadj-Alouane A B 1992 *A Dual Genetic Algorithm for Bounded Integer Programs* TR 92-53 Department of Industrial and Operations Engineering, The University of Michigan

Bilchev G 1995 Private communication

Bilchev G and Parmee I C 1995 *Ant Colony Search vs. Genetic Algorithms* Technical Report, Plymouth Engineering Design Centre, University of Plymouth

Colorni A, Dorigo M and Maniezzo V 1991 Distributed optimization by ant colonies *Proc. Eur. Conf. Artificial Life (December 1991, Paris)*

De Jong K A 1975 *An Analysis of the Behavior of a Class of Genetic Adaptive Systems* Doctoral Dissertation, University of Michigan *Dissertation Abstract Int.* **36** 5140B (University Microfilms 76-9381)

Eshelman L J and Schaffer J D 1993 Real-coded genetic algorithms and interval schemata *Foundations of Genetic Algorithms* vol 2 (Los Altos, CA: Morgan Kaufmann) pp 187–202

Floudas C A and Pardalos P M 1987 *A Collection of Test Problems for Constrained Global Optimization Algorithms (Lecture Notes in Computer Science 455)* (Berlin: Springer)

Fogel D B and Stayton L C 1994 On the effectiveness of crossover in simulated evolutionary optimization *BioSystems* **32** 171–82

Hadj-Alouane A B and Bean J C 1992 *A Genetic Algorithm for the Multiple-Choice Integer Program* TR 92-50 Department of Industrial and Operations Engineering, The University of Michigan

Hock W and Schittkowski K 1981 *Test Examples for Nonlinear Programming Codes (Lecture Notes in Economics and Mathematical Systems 187)* (Berlin: Springer)

Homaifar A, Lai S H-Y and Qi X 1994 Constrained optimization via genetic algorithms *Simulation* **62** 242–54

Joines J A and Houck C R 1994 On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with GAs *Proc. IEEE ICEC 1994* pp 579–84

Keane A 1994 *Genetic Algorithms Digest* **8** issue 16

Le Riche R G, Knopf-Lenoir C, and Haftka R T 1995 A segregated genetic algorithm for constrained structural optimization *Proc. 6th Int. Conf. Genetic Algorithms (July 1993, Urbana-Champaign)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 558–65

Michalewicz Z 1995 Genetic algorithms, numerical optimization and constraints *Proc. 6th Int. Conf. Genetic Algorithms (July 1993, Urbana-Champaign)* ed L Eshelman (San Mateo, CA: Morgan Kaufmann) pp 151–8

——1996 *Genetic Algorithms + Data Structures = Evolution Programs* 3rd edn (New York: Springer)

Michalewicz Z and Attia N 1994 Evolutionary optimization of constrained problems *Proc. 3rd Ann. Conf. on Evolutionary Programming (February 1994, San Diego, CA)* (Singapore: World Scientific) pp 98–108

Michalewicz Z and Nazhiyath G 1995 Genocop III: a co-evolutionary algorithm for numerical optimization problems with nonlinear constraints *Proc. 2nd IEEE Int. Conf. on Evolutionary Computation (November–December 1995, Perth)* pp 647–51

Orvosh D and Davis L 1993 Shall we repair? Genetic algorithms, combinatorial optimization, and feasibility constraints *Proc. 5th Int. Conf. on Genetic Algorithms (July 1993, Urbana-Champaign)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) p 650

Paredis J 1994 Co-evolutionary Constraint Satisfaction *Proc. 3rd Conf. on Parallel Problem Solving from Nature (October 1994, Jerusalem)* (Berlin: Springer) pp 46–55

Powell D and Skolnick M M 1993 Using genetic algorithms in engineering design optimization with non-linear constraints *Proc. 5th Int. Conf. Genetic Algorithms (July 1993, Urbana-Champaign)* (San Mateo, CA: Morgan Kaufmann) pp 424–30

Reynolds R G 1994 An introduction to cultural algorithms *Proc. Third Ann. Conf. on Evolutionary Programming (February 1994, San Diego, CA)* ed A V Sebald and L J Fogel (River Edge, NJ: World Scientific) pp 131–9

Reynolds R G, Michalewicz Z and Cavaretta M 1995 Using cultural algorithms for constraint handling in Genocop *Proc. 4th Ann. Conf. on Evolutionary Programming (March 1995, San Diego, CA)* (San Diego, CA, 1995) ed J R McDonnell, R G Reynolds and D B Fogel, pp 289–305

Richardson J T, Palmer M R, Liepins G and Hilliard M 1989 Some guidelines for genetic algorithms with penalty functions *Proc. 3rd Int. Conf. Genetic Algorithms* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 191–7

Schoenauer M and Xanthakis S 1993 Constrained GA optimization *Proc. 5th Int. Conf. Genetic Algorithms (June 1995, George Mason University)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 573–80

Surry P D, Radcliffe N J and Boyd I D 1995 A multi-objective approach to constrained optimization of gas supply networks *AISB-95 Workshop on Evolutionary Computing (April 1995, Sheffield) (Sheffield, 1995)*

Smith A and Tate D 1993 Genetic optimization using a penalty function *Proc. 5th Int. Conf. Genetic Algorithms (July 1993, Urbana-Champaign)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 499–503

Wright A H 1991 Genetic algorithms for real parameter optimization *1st Workshop on the Foundations of Genetic Algorithms and Classifier Systems* (San Mateo, CA: Morgan Kaufmann) ed G Rawlins pp 205–18

G9.10 Quadratic assignment

Volker Nissen

Abstract

The quadratic assignment problem (QAP) is of practical relevance in different fields of application, such as hospital layout planning, machine scheduling, and component placement on printed circuit cards. We discuss the QAP as a base model for facility layout problems and present an evolutionary solution technique derived from evolution strategy (ES), one of the mainstream forms of evolutionary algorithms (EAs). It is empirically evaluated on a test suite of QAPs. The implemented ES for combinatorial problems (CES) is deliberately not hybridized with other solution techniques. This gives an idea of the potential of ES on this standard problem in the light of recent debates about the competitiveness of EAs for solving combinatorial problems. CES is a good heuristic for solving QAPs and does not require tuning of strategy parameters on individual problem instances. Even though CES is superior over the classical 2-Opt procedure and an evolutionary approach proposed by Tate and Smith (1995), it is not fully competitive with Threshold Accepting, one of the most efficient heuristics currently available for QAPs. Implications of these results are discussed.

G9.10.1 Facility layout and quadratic assignment

Locating facilities with material flow between them, such as machines in a factory hall, is a difficult layout problem that is frequently modeled as a quadratic assignment problem (QAP) (Kusiak and Heragu 1987). The QAP is one of the most difficult combinatorial optimization problems and can be formalized as follows (Burkard 1990): given a set $N = \{1, 2, \ldots, n\}$ and real numbers c_{ik}, a_{ik}, b_{ik} for $i, k = 1, 2, \ldots, n$, find a permutation φ of the set N which minimizes

$$Z = \sum_{i=1}^{n} c_{i\varphi(i)} + \sum_{i=1}^{n}\sum_{k=1}^{n} a_{ik} b_{\varphi(i),\varphi(k)} \qquad \text{(G9.10.1)}$$

where n is the total number of facilities and locations, c_{ij} is the fixed cost of locating facility j at location i, b_{jl} is the flow of material from facility j to facility λ, and a_{ik} is the cost of transferring a material unit from location i to location k.

The linear part in (G9.10.1) may be considered as installation costs while the quadratic part accounts for interaction (material traffic) between facilities. The QAP is characterized by a high degree of interaction between solution elements (assignments). Even swapping the assignment of two facilities might affect the quality of virtually all other assignments, depending on the flow matrix. As a generalization of the *traveling salesman problem* (TSP), the QAP is NP-hard, and only moderately sized problem instances ($n \simeq 18$) can be solved to optimality with exact algorithms within reasonable time limits. One therefore concentrates on developing heuristics for the QAP. Extensive reviews of the QAP and associated solution techniques can be found in the articles by Kusiak and Heragu (1987) and Burkard (1990).

G9.5

Vollmann and Buffa (1966) introduced the concept of flow dominance, measuring the variation of values in the flow matrix. It is given by

$$100\,\text{std dev.}/\text{mean}$$

of the matrix elements. Simply stated, high flow dominance indicates that a few facilities with high interaction tend to dominate the problem. Burkard and Fincke (1983) were the first to prove the asymptotic behavior of large randomly generated QAPs. This means that the relative difference between the worst and the optimal solution becomes arbitrarily small with a probability tending to unity as the problem size tends to infinity. Thus, instead of focusing only on the large random problems frequently cited in the literature, it is more appropriate to use a suite of test problems varying in size and structure, where flow dominance is one sensible measure to characterize the structure of a QAP. Here, a set of seven QAPs varying in size between $n = 15$ and $n = 64$ with different structure (flow dominance) has been employed. The problems were taken from the QAP library collected by Burkard *et al* (1991).

A number of authors have previously developed evolutionary approaches to solve QAPs (see overviews by Alander (1995) and Nissen (1995)). When high-quality results were achieved, the EA frequently had been hybridized with other well-known problem solving techniques such as simulated annealing or tabu search. It is, therefore, difficult to assess the potential of the evolutionary part of these heuristics for QAPs.

In this case study, a variant of *evolution strategy* (ES) is proposed to solve QAPs. Since ES was originally not invented for combinatorial optimization, the methodology was adapted to suit the needs of this application while staying in the evolutionary framework. However, solving the QAP can only be considered a first step to approaching real-world facility layout problems that frequently involve additional complex constraints. For instance, locations may be of unequal size, permitting only a subset of machines to be located at a certain position. Fixed costs for material transport can occur. Safety or technical considerations may yield certain assignments invalid. Accurate material flow data may not be available. Locations might not be determined in advance. In a *multicriterion decision* situation, aspects such as safety or flexibility of a layout as well as environmental objectives might be additional goals. However, only a few authors (e.g. Tam 1992, Kouvelis *et al* 1992, Smith and Tate 1993, Krause and Nissen 1995) have extended the QAP to include some such practically relevant considerations.

B1.3

C4.5, F1.9

G9.10.2 Design and implementation of the evolution strategy

Our ES variant for combinatorial problems, termed CES, was first presented by Nissen (1994b). It uses a straightforward permutation coding (figure G9.10.1). A simple population concept, basically a $(1, \lambda)$ ES, is employed. This means, in each generation λ offspring are generated from one parent solution. First, one produces λ copies of the parent. Then, each of these copies is mutated by (possibly repeated) pairwise exchange of randomly determined positions of facilities (assignments) on the given solution, thereby generating a population of offspring. Note that the mutation operator from standard ES, which is based on normally distributed random variables, is not adequate for such a permutation coding since it would yield invalid results. Crossover of solutions is not employed. $\lambda = 50$ for the smaller instances NUG15, NUG20 and ELS19. For the other problems $\lambda = 100$. The parent is eliminated after each generation.

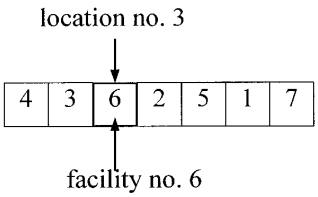


Figure G9.10.1. Representation of a solution for $n = 7$. A similar figure appeared in Nissen (1994b) (copyright 1994 IEEE).

The number of pairwise exchanges during mutation is restricted to be randomly either one or two. It can occasionally be zero, however, should the algorithm by chance choose the same position for a swap twice. Too many exchanges would generally deteriorate the objective function value of a given QAP solution due to the massive interactions (traffic) between the facilities.

The best offspring becomes the new parent. If the parent's objective function value represents no improvement over the former parent, a counter is increased. The counter is reset to zero whenever a CES generation is successful, that is, an improvement is achieved. After a certain number of consecutive

unsuccessful generations g_u, a procedure called destabilization is executed. It was found empirically that $(n/10 + 2)$, where the result is rounded, is a good value for g_u. This parameter value was used in all experiments but no claim is made as to its optimality.

Destabilization is essentially a more intensive form of mutation. During this phase, the counter is set to zero and λ offspring are created with increased mutation intensity. The number of swaps now randomly lies in the interval $[3, \ldots, 8]$. Thereby, individuals which differ more strongly from previous solutions are generated and the search shifts to a new area in the solution space. This helps to escape from local optima and counters the strong selection pressure in CES. Again, the best offspring is determined to become the new parent. Procedure destabilization is then terminated and the search continues as before until t_{max}, the maximum number of generations.

CES starts from a randomly generated initial solution. The best solution ever found by the heuristic is stored separately, and it is continuously updated during the search. This is the final result of the heuristic.

An overview of CES is given in the following pseudocode. The parameters were empirically determined and kept constant over all QAP experiments with the exception of the loop variable i that runs from 1 to 50 for the smaller problems NUG15, NUG20 and ELS19.

```
Input:      t_max
Output:     x*, the best solution ever found.
1   t ← 0;
2   create initial solution x_0 randomly;
3   x* ← x_0;
4   fail_count ← 0;
5   for t ← 1 to t_max do
6       f̃ ← f(x_0);
7       for i ← 1 to 100 do
8           x_i ← x_0;   {copy parent}
9           sample no_swaps U(1, 2);
10          mutate x_i by swapping assignments according to no_swaps;
            {details see main text}
11          evaluate f(x_i);
        od
12      x_0 ← x_j where f(x_j) = min{f(x_i) |i = 1, ..., 100};   {selection}
13      if (f(x_0) ≥ f̃)
            then increment fail_count;
            else fail_count← 0;
                 check if update of x* necessary;
        fi
14      if (fail_count = round(n/10) + 2)
            then destabilization;
    od
15  output x*;
{procedure destabilization}
1   destabilization (x_0)
2       for i ← 1 to 100 do
3           x_i ← x_0;   {copy parent}
4           sample no_swaps ~ U(3, 8);
5           mutate x_i by swapping assignments according to no_swaps;
            {details see main text}
6           evaluate f(x_i);
        od
7       x_0 ← x_j where f(x_j) = min {f(x_i) |i = 1, ..., 100};   {selection}
8       check if update of x* necessary;
9       fail_count← 0;
    return (x_0);
```

G9.10.3 Empirical results

G9.10.3.1 Performance of CES on the test suite

CES was run on seven test problems originally published by Nugent *et al* (1968) (NUG15, NUG20, NUG30), Steinberg (1961) (STE36a, STE36c), Elshafei (1977) (ELS19), and Skorin-Kapov (1990) (SKO64) with numbers of locations and facilities n (including dummy facilities) varying between 15 and 64. NUG15, NUG20, NUG30, and SKO64 are randomly generated problems with low flow dominance. The other three appear to be practical applications with high (STE36a, STE36c) and very high (ELS19) flow dominance values. The search space size (number of solution alternatives) varied from roughly 1.31×10^{12} for NUG15 to 1.27×10^{89} for SKO64. CES was implemented in Pascal on a workstation IBM RS 6000/320. Ten runs were performed in each experiment. It should be noted that in the work of Nissen and Paul (1995) we alternatively used five, ten and 30 initial solutions (=different runs) for evaluating another QAP heuristic. The performance measures, mean and standard deviation (std dev.) of the best objective function values from different runs, were apparently unaffected by this choice of the number of runs. Results for CES are given in table G9.10.1. Data for generation 0 refer to initial solutions.

A typical convergence chart for CES appears in figure G9.10.2. Generally, the optimal solution was approached in an asymptotic manner. Improvements were a little less continuous on ELS19, the problem with the highest flow dominance value. When flow dominance is very high, heuristics based on pairwise exchanges of assignments have difficulties in overcoming the pronounced local suboptima. Crossover can be advantageous in such a case, as was found in an empirical investigation on QAPs involving genetic algorithms (Nissen 1994a).

However, due to the chosen search operator based on pair exchanges CES allows for an efficient form of solution evaluation that could not be applied when crossover was used (see Nissen 1994a for details).

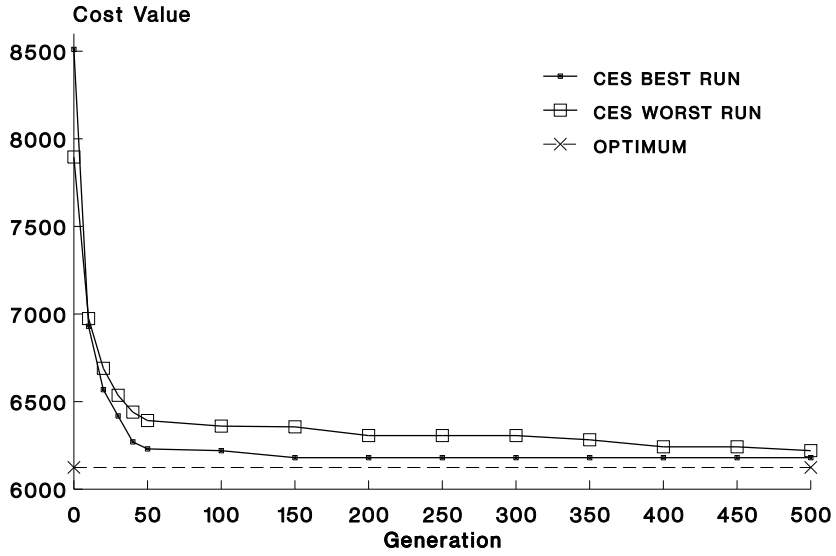


Figure G9.10.2. A typical convergence chart for CES: the best and worst of ten runs on NUG30. A similar figure appeared in Nissen (1994b) (copyright 1994 IEEE).

CES identified good solutions on all seven problem instances. Destabilization proved to be a useful heuristic element. On larger QAPs, there was a tendency for fewer destabilization phases. One reason is the way the allowed number of consecutive unsuccessful CES generations before destabilization is computed. The larger n, the higher this maximum value. Besides, with increasing problem dimension, it becomes easier to leave a local optimum. However, because the size of the search space ($n!$) rises drastically, more time is needed to identify high-quality solutions as compared to smaller problem instances.

CES is quite a useful heuristic for solving QAPs. It has acceptable CPU requirements, is easily implementable, and yields good results without problem-specific parameter tuning on QAPs of very different sizes and structures.

Table G9.10.1. CES results, starting from random initial solutions. Mean OFV is the mean objective function value of the best solution found up to this generation, averaged over ten runs on an IBM RS 6000/320. AFE is the average total number of function evaluations per run up to this generation.

Test problem Best known solution	Generation	Mean OFV	AFE	Avg. CPU (s)
NUG15	0	1 564	1	
1150	200	1 162	10 811	1.5
	1 000	1 153	54 446	7.4
	2 000	1 151	108 976	14.7
	6 000	1 150	327 086	44.2
NUG20	0	3 442	1	
2570	200	2 626	10 641	2.1
	1 000	2 592	53 276	10.5
	2 000	2 587	106 626	21.0
	6 000	2 574	320 096	62.9
	20 000	2 570	1 067 451	209.6
NUG30	0	8 127	1	
6124	100	6 310	10 291	3.6
	500	6 202	52 281	18.5
	1 000	6 166	104 701	37.0
	3 000	6 145	314 691	111.4
	10 000	6 135	1 050 181	372.4
SKO64	0	58 947	1	
48498	100	50 196	10 001	19.1
	500	49 565	50 271	95.9
	1 000	49 380	100 781	192.3
	3 000	49 044	302 591	583.1
	10 000	48 906	1 009 211	1927.5
ELS19	0	59 725 819	1	
17212548	200	19 218 337	10 486	1.9
	1 000	18 128 394	52 916	9.7
	2 000	18 128 394	105 851	19.4
	6 000	17 517 830	317 621	58.2
STE36a	0	22 755	1	
9526	100	10 650	10 171	4.8
	500	10 103	51 461	24.4
	1 000	9 979	102 981	48.8
	3 000	9 812	309 561	146.8
	10 000	9 701	1 032 511	489.2
STE36c	0	18 890	1	
8239.1	100	8 923	10 131	4.8
	500	8 713	51 301	24.4
	1 000	8 582	102 901	49.1
	3 000	8 403	309 141	147.6
	10 000	8 338	1 031 451	492.3

G9.10.3.2 Comparison of CES with other approaches

The well known traditional combinatorial heuristic 2-Opt served as a first benchmark to compare the quality of solutions generated by CES with some other QAP heuristics (tables G9.10.2 and G9.10.3). 2-Opt is a simple local search heuristic that sequentially considers pairwise exchange between the positions of facilities. The swap is made whenever this results in a lower objective function value and the search starts again from the new solution. This procedure continues until no exchange of assignments in the current solution results in a further improvement. 2-Opt was implemented in Pascal on an IBM RS 6000/320, as was CES. The initial solutions of 2-Opt in table G9.10.2 were identical to those of CES.

CES on average quickly produced far better solutions than 2-Opt. Even after only 100 generations CES often generated better mean values. Moreover, it converged to better solutions with greater reliability as shown by the small std dev. in table G9.10.3.

Tate and Smith (1995) also proposed an evolutionary heuristic for solving QAPs. While they termed

Table G9.10.2. 2-Opt results with initial solutions identical to CES. Mean OFV is the mean objective function value of the best solution found, averaged over ten runs on an IBM RS 6000/320. AFE is the average total number of function evaluations per run.

Test problem	Mean OFV	AFE	Avg. CPU (s)
NUG15	1 194	746	0.1
NUG20	2 671	2 002	0.2
NUG30	6 322	10 322	2.2
SKO64	49 524	188 650	241.6
ELS19	21 798 726	3 594	0.4
STE36a	10 447	17 839	5.2
STE36c	8 903	20 188	5.9

Table G9.10.3. A comparison of CES and 2-Opt on NUG30 with respect to solution quality at approximately identical CPU requirements. All initial solutions were generated randomly. A similar table appeared in Nissen (1994b) (copyright 1994 IEEE).

$10 \times$ CES (10 000 generations)				$1700 \times$ 2-Opt			
Best	Worst	Mean	Std dev.	Best	Worst	Mean	Std dev.
6124	6150	6135	9.3	6128	6702	6351	83.9

Table G9.10.4. Results of the evolutionary heuristic (25% crossover, 75% mutation) presented by Tate and Smith (1995) for test problems also used here. Mean OFV is the mean objective function value of the best solution found, averaged over ten runs. (Original values are doubled to account for symmetrical flow.)

Test problem	Mean OFV	Funct. eval. per run
NUG15	1170	$\approx$ 200 000
NUG20	2643	200 000
NUG30	6305	200 000

it a genetic algorithm, it is quite close to an evolution strategy, using some sort of $(\mu + \lambda)$-selection and focusing on mutation rather than crossover. Some results with this approach for test problems also used here are reported in table G9.10.4. Since different hardware and software was used in the implementation of CES and the heuristic by Tate and Smith, we do not mention CPU times but function evaluations to account for computational effort. CPU requirements are primarily influenced by the calculation of objective function values in this application.

Let the efficiency of a search procedure be defined as the ratio of solution quality and required search effort. CES clearly showed a better performance than the other heuristic in terms of efficiency, averaged over ten runs. Moreover, Tate and Smith estimated the increase in computational effort per solution generated to be quadratic with the number of sites for their heuristic, which is higher than for CES and other approaches based on pairwise exchange of assignments, such as tabu search (see e.g. Fleurent and Ferland 1994).

Threshold Accepting (TA) is a local search technique and a much stronger competitor than 2-Opt or the evolutionary heuristic by Tate and Smith. TA is a simplification of the well known simulated annealing procedure that was initially proposed by Dueck and Scheuer (1990). Starting from an initial solution each TA step consists of a slight change of the old solution into a new one. Then, the qualities of the two solutions are compared with respect to the given objective function. TA accepts every solution that is either better than the current solution or that deteriorates the old objective function value by less than a given threshold level T. The new solution then replaces the old solution as a basis for the next TA step. The threshold T will be relatively large at the beginning of the search process to allow for a full exploration of the solution space. As the search continues, T is lowered in a stepwise manner. Generally, an increasing number of trials is performed at successive levels since lower thresholds will expand the time required to

reach some form of equilibrium or ground state. The search process terminates when a minimum threshold level is reached. Nissen and Paul (1995) modified the basic TA heuristic in several ways and applied it to the QAP. In this implementation, new solutions were obtained from a given configuration by a simple random pair exchange of assignments. The modified TA scheme appears to be one of the most efficient heuristics currently available for the QAP.

Table G9.10.5. Comparison of CES and TA for various run lengths. Data for TA was partly taken from Nissen and Paul (1995). Since hardware was different not CPU time but function evaluations are reported to compare efficiency. Results are based on 10 runs each with random initial solutions. AFE = average total number of function evaluations per run. For CES and TA, the total number of evaluations varied only very slightly between different runs on the same test problem.

Test problem	CES			TA		
Best known solution	Best	Mean	AFE	Best	Mean	AFE
NUG20				2 614	2 649	1 876
2570	2 580	2 626	10 641	2 570	2 604	17 787
NUG	2 570	2 587	106 626	2 570	2 585	72 002
6124	6 220	6 310	10 291	6 230	6 330	3 458
	6 150	6 202	52 281	6 128	6 179	35 997
SKO64	6 128	6 145	314 691	6 124	6 148	225 157
48498	49 720	50 196	10 001	49 340	49 754	9 206
	48 872	49 044	302 591	48 602	48 919	121 263
ELS19	48 778	48 906	1 009 211	48 550	48 747	764 803
17212548	17 212 548	19 218 337	10 486	17 997 928	21 593 135	6 036
	17 212 548	18 128 394	52 916	17 937 024	21 404 362	52 478
	17 212 548	17 517 830	317 621	17 937 024	18 684 375	269 592
STE36a	10 218	10 650	10 171	9 972	10 245	8 854
9526	9 798	9 979	102 981	9 562	9 834	97 359
	9 564	9 701	1 032 511	9 562	9 615	516 421

In table G9.11.5, CES and TA are compared in terms of their efficiency. Again, the comparison of computational requirements is based on function evaluations and not CPU time for the same reasons as before. The rise in computational effort per solution generated is roughly comparable for CES and TA as the problem size increases.

TA was superior over CES in all cases but ELS19 with its particularly high flow dominance and comparatively low dimensionality. This instance is difficult for any heuristic that only relies on a simple pair exchange of assignments. One could probably improve the performance of TA on this problem by introducing a destabilization (as in CES) or by transferring the idea of crossover from genetic algorithms. Even though CES is a good heuristic, one must conclude that it is not fully competitive with the best solution techniques currently available for the QAP.

G9.10.4 Conclusions

The variant of evolution strategy for combinatorial problems (CES) proposed here compared favorably with the classic 2-Opt procedure and results of an evolutionary heuristic by Tate and Smith on the quadratic assignment problem. The destabilization operator in our ES implementation is useful in overcoming local optima when selection pressure is high, as in CES. It is also successful on problem instances with high flow dominance that prove difficult for heuristics purely based on pairwise exchange of assignments. CES is a good heuristic for the QAP that, moreover, requires no tuning of strategy parameters on individual problem instances, making it user friendly. A more detailed description of the experimental setup, further experiments with genetic algorithms and evolutionary programming, and results of sensitivity analysis are given by Nissen (1994a).

CES could not fully compete with Threshold Accepting, however, one of the most efficient heuristics currently available for this application. This might fuel the debate on the potential of EAs in combinatorial optimization. However, one should recall that in our experiments hybridizing with other solution techniques was deliberately avoided. Fleurent and Ferland (1994), for instance, developed such hybrids for the QAP.

They successfully combined genetic algorithms with tabu search to produce competitive optimization techniques that were able to improve upon the best known solutions for a number of large random QAP test problems. However, the CPU requirements of their hybrids to achieve these improvements were very high (up to a few days on a SPARC 10 for some problem instances). One gets the feeling that, at least for the QAP, the competitiveness of the evolutionary approach to other modern heuristics actually depends on whether CPU requirements are of importance or parallel hardware is available, respectively. In facility layout, one can usually ignore CPU time so that carefully hybridized EAs can be competitive solution techniques.

References

Alander J T 1995 *An Indexed Bibliography of Genetic Algorithms in Operations Research* Report Series no 94-1-OR, Department of Information Technology and Production Economics, University of Vaasa

Burkard R E 1990 Locations with spatial interactions: the quadratic assignment problem *Discrete Location Theory* ed P B Mirchandani and R L Francis (New York: Wiley) pp 387–437

Burkard R E and Fincke U 1983 The asymptotic probabilistic behaviour of quadratic sum assignment problems *Z. Operat. Res.* **27** 73–81

Burkard R E, Karisch S and Rendl F 1991 QAPLIB—a quadratic assignment problem library *Eur. J. Operat. Res.* **55** 115–9

Dueck R E and Scheuer T 1990 Threshold accepting: a general purpose optimization algorithm appearing superior to simulated annealing *J. Comput. Phys.* **90** 161–75

Elshafei A N 1977 Hospital layout as a quadratic assignment problem *Operat. Res. Q.* **28** 167–79

Fleurent C and Ferland J A 1994 Genetic hybrids for the quadratic assigment problem *DIMACS Series in Discrete Mathematics and Theoretical Computer Science 16* ed P M Pardalos and H Wolkowicz (Providence, RI: AMS) pp 173–88

Kouvelis P, Chiang W-C and Fitzsimmons J 1992 Simulated annealing for machine layout problems in the presence of zoning constraints *Eur. J. Operat. Res.* **57** 203–23

Krause M and Nissen V 1995 On using penalty functions and multicriteria optimisation techniques in facility layout *Evolutionary Algorithms in Management Applications* ed J Biethahn and V Nissen (Berlin: Springer) pp 153–66

Kusiak A and Heragu S S 1987 The facility layout problem *Eur. J. Operat. Res.* **29** 229–51

Nissen V 1994a *Evolutionäre Algorithmen Darstellung, Beispiele, betriebswirtschaftliche Anwendungsmöglichkeiten.* (Wiesbaden: DUV)

——1994b Solving the quadratic assignment problem with clues from nature *IEEE Trans. Neural Networks: Special Issue on Evolutionary Programming* **NN-5** 66–72

——1995 An overview of evolutionary algorithms in management applications *Evolutionary Algorithms in Management Applications* ed J Biethahn and V Nissen (Berlin: Springer) pp 44–98

Nissen V and Paul H 1995 A modification of threshold accepting and its application to the quadratic assignment problem *OR Spektrum* **17** 205–10

Nugent E N, Vollmann T E and Ruml J 1968 An experimental comparison of techniques for the assignment of facilities to locations *Operat. Res.* **16** 150–73

Skorin-Kapov J 1990 Tabu search applied to the quadratic assignment problem *ORSA J. Comput.* **2** 33–45

Smith A E and Tate D M 1993 Genetic optimization using a penalty function *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, July 1993* ed S Forrest (San Mateo, CA: Morgan-Kaufmann) pp 499–503

Steinberg L 1961 The backboard wiring problem *SIAM Rev.* **3** 37–50

Tam K Y 1992 Genetic algorithms, function optimization, and facility layout design *Eur. J. Operat. Res.* **63** 322–46

Tate D M and Smith A E 1995 A genetic approach to the quadratic assignment problem *Comput. Operat. Res.* **22** 73–83

Vollmann T E and Buffa E S 1966 The facility layout problem in perspective *Management Sci.* B **12** 450–68

H1.1 Future work and practical applications of genetic programming

John R Koza

Abstract

Genetic programming is a relatively new domain-independent method for evolving computer programs to solve problems. This section suggests avenues for possible future research on genetic programming, opportunities to extend the technique, and areas for possible practical applications.

H1.1.1 Introduction

The goal of the field of automatic programming is to create, in an automated way, a computer program that enables a computer to solve a problem. *Genetic programming* (Koza 1992, 1994) is a domain-independent approach to automatic programming in which computer programs are evolved to solve, or approximately solve, problems. The field of genetic programming has grown rapidly in the past few years. Between 1992 and 1996, over 600 papers on genetic programming have been published. B1.5.1

This paper discusses the many opportunities to apply genetic programming to realistic and practical problems, numerous possible avenues to extend the technique of genetic programming, and avenues for research on theoretical aspects of genetic programming.

H1.1.2 Promising application areas

I believe the single most important area for future work in genetic programming (as well as for all other techniques of automated machine learning) is to demonstrate the applicability of the technique to realistic problems.

The presence of some or all of the following characteristics make an area especially suitable for the application of genetic programming:

- an area where conventional mathematical analysis does not, or cannot, provide analytic solutions
- an area where the interrelationships among the relevant variables are poorly understood (or where it is suspected that the current understanding may well be wrong)
- an area where finding the size and shape of the ultimate solution to the problem is a major part of the problem
- an area where an approximate solution is acceptable (or is the only result that is ever likely to be obtained)
- an area where there are a large number of data, in computer readable form, that require examination, classification, and integration
- an area where small improvements in performance are routinely measured (or easily measurable) and highly prized.

For example, problems in *automated control* are especially well suited for genetic programming because of the inability of conventional mathematical analysis to provide analytic solutions to many problems of practical interest, the willingness of control engineers to accept approximate solutions, and the high value placed on small incremental improvements in performance. F1.3

Problems in fields where large quantities of data are accumulating in machine readable form (e.g. biological sequence data, astronomical observations, geological and petroleum data, financial time series data, satellite observation data, weather data, news stories, and marketing databases) also constitute especially interesting areas for potential practical applications of genetic programming.

H1.1.3 Practical applications

H1.1.3.1 The threshold of practicality

Evidence is accumulating that genetic programming is now reaching the threshold of delivering results that are competitive with human performance on nontrivial problems. There have been several recent examples of problems—from fields as diverse as cellular automata, space satellite control, molecular biology, and design of electrical circuits—in which genetic programming has evolved a computer program whose results were, under some reasonable interpretation, competitive with human performance on the specific problem. For example, genetic programming with automatically defined functions has evolved a rule for the majority classification task for one-dimensional two-state cellular automata with an accuracy that exceeds that of the original human-written Gacs–Kurdyumov–Levin (GKL) rule, all other known subsequent human-written rules, and all other known rules produced by automated approaches for this problem (Andre *et al* 1996). Another example involves the near-minimum-time control of a spacecraft's attitude maneuvers using genetic programming (Howley 1996). A third example involves the discovery by genetic programming of a computer program to classify a given protein segment as being a *transmembrane domain* without using biochemical knowledge concerning hydrophobicity (Koza 1994, Koza and Andre 1996a, b). A fourth example illustrated how automated methods may prove to be useful in discovering biologically meaningful information hidden in the rapidly growing databases of DNA sequences and protein sequences. Genetic programming successfully evolved motifs for detecting the D-E-A-D box family of proteins and for detecting the manganese superoxide dismutase family that detected the two families either as well as, or slightly better than, the comparable human-written motifs found in the database created by an international committee of experts on molecular biology (Koza and Andre 1996c). A fifth example involves the design of difficult-to-design electrical circuits using genetic programming (Koza *et al* 1996). A sixth example is recent work on facility layouts (Garces-Perez *et al* 1996). G6.1

H1.1.3.2 Handling complex data structures

Ordinary computer programs use numerous well known techniques for handling vectors of data, arrays, and more complex data structures. One important area for work on technique extensions for genetic programming involves developing workable and efficient ways to handle vectors, arrays, trees, graphs, and more complex data structures. Such new techniques would have immediate application to a number of problems in such fields as computer vision, biological sequence analysis, economic time series analysis, and pattern recognition where a solution to the problem involves analyzing the character of an entire data structure. Recent work in this area includes that of Langdon (1996) in handling more complex data structures, Teller (1996) in understanding images represented by large arrays of pixels, and Handley (1996) in applying statistical computing zones to biological sequence data.

H1.1.3.3 Evolution of mental models

Complex adaptive systems usually possess a mechanism for modeling their environment. A mental model of the environment enables a system to contemplate the effects of future actions and to choose an action that best fulfills its goal. Brave (1996b) has developed a special form of memory that is capable of creating relations among objects and then using these relations to guide the decisions of a system.

H1.1.3.4 Evolution of assembly code

The innovative work by Nordin (1994) in developing a version of *genetic programming* in which the programs are composed of sequences of low-level machine code offers numerous possibilities for extending the techniques of genetic programming (especially for programs with loops) as well as enormous savings in computer time. These savings can then be used to increase the scale of problems being considered. G1.5

H1.1.3.5 Automatically defined functions and macros

Computer programs gain leverage in solving complex problems by means of reusable and parametrizable subprograms. Automated machine learning can become scalable (and truly useful) only if there are techniques for creating large and complex problem-solving programs from smaller building blocks. Rosca (1995) has analyzed the workings of hierarchical arrangements of subprograms in genetic programming. Spector (1996) has developed the notion of automatically defined macros (ADMs) for use in evolving control structures. Considerable future work can be anticipated in this area.

H1.1.3.6 Cellular encoding

Gruau (1994) described an innovative technique, called *cellular encoding* or *developmental genetic programming*, in which genetic programming is used to concurrently evolve the architecture of a neural network, along with the weights, thresholds, and biases of the individual neurons in the neural network. In this technique, each individual program tree in the population is a specification for developing a complete neural network from a starting point consisting of a very simple embryonic neural network containing a single neuron. Genetic programming is applied to populations of these network-constructing program trees in order to evolve a neural network to solve various problems. Brave (1996a) has extended and applied this technique to the evolution of finite automata. This technique has also been applied to other complex structures, such as electrical circuits (Koza *et al* 1996).

H1.1.3.7 Automatic programming of multiagent systems

The cooperative behavior of multiple independent agents can potentially be harnessed to solve a wide variety of practical problems. However, programming of multiagent systems is particularly vexatious. Bennett's recent work (1996) in evolving the number of independent agents while concurrently evolving the specific behaviors of each agent and the recent work by Luke and Spector (1996) in evolving teamwork are opening this area to the application of genetic programming.

H1.1.3.8 Autoparallelization of algorithms

The problem of mapping a given sequential algorithm onto a parallel machine is usually more difficult than writing a parallel algorithm from scratch. The recent work of Walsh and Ryan (1996) is advancing the autoparallelization of algorithms using genetic programming. Considerable future work can be anticipated in this important area.

H1.1.3.9 Coevolution

In nature, individuals do not evolve in a vacuum. Instead, there is coevolution that involves interactions between agents and other agents as well as between agents and their physical environment. The important area of coevolution, as illustrated by the work of Pollack and Blair (1996), can be expected to attract considerable future work.

H1.1.3.10 Complex adaptive systems

Genetic programming has proven useful in evolving complex systems, such as Lindenmayer systems (Jacob 1996) and cellular automata (Andre *et al* 1996) and can be expected to continue to be useful in this area.

H1.1.3.11 Evolution of structure

One of the most vexatious aspects of automated machine learning from the earliest times has been the requirement that the human user predetermine the size and shape of the ultimate solution to his problem (Samuel 1959). There can be expected to be continuing research on ways by which the size and shape of the solution can be made part of the *answer* provided by the automated machine learning technique, rather than part of the *question* supplied by the human user. For example, architecture-altering operations (Koza 1995) enable genetic programming to introduce (or delete) function-defining branches, to adjust the number of arguments of each function-defining branch, and to alter the hierarchical references among function-defining branches. Brave (1995) showed that recursion could be implemented within genetic programming. Future work can be expected on operations that enable genetic programming to dynamically introduce iteration and recursion and nested occurrences of iteration and recursion.

H1.1.3.12 Foundations of genetic programming

Genetic programming inherits many of the mathematical and theoretical underpinnings from John Holland's pioneering work (1975) in the field, including the near-optimality of Darwinian search. However, the genetic algorithm is a dynamical system of extremely high dimensionality. Many of the most basic questions about the operation of the algorithm and the domain of its applicability are only partially understood. The transition from the fixed-length character strings of the genetic algorithm to the variable-sized Turing-complete program trees (and even program graphs) of genetic programming further compounds the difficulty of the theoretical issues involved. There is increasing work on the grammatical structure of genetic programming (Whigham 1996).

H1.1.3.13 Optimization

The fundamental importance of optimization problems guarantees that there will be considerable future work on applying genetic programming to optimization. Recent examples include work (Soule *et al* 1996) from the University of Idaho, the site of much early work on genetic programming techniques, and the work of Garces-Perez and coworkers (1996).

H1.1.3.14 Novel methods of fitness evaluation

In a novel experiment, Floreano and Mondada (1994) ran the genetic algorithm on a fast workstation to evolve a control strategy for an *obstacle-avoiding robot*. The fitness of an individual strategy in the population within a particular generation of the run was determined by executing that strategy on a physical robot tethered to the workstation for 30 seconds in real time. The robot behavior is thus highly realistic and avoids the pitfalls of computer-simulated behavior. This technique can be expected to find future application in genetic programming.

G3.6, G3.7

H1.1.3.15 Techniques that exploit parallel hardware

Evolutionary algorithms offer the ability to solve problems in a domain-independent way that requires little domain-specific knowledge. However, the price of this domain independence and knowledge independence is paid in execution time. Application of genetic programming to realistic problems inevitably requires considerable horsepower. The long-term trend toward ever faster microprocessors is likely to continue to provide ever increasing amounts of computational power. However, for those using algorithms that can beneficially exploit parallelization (such as genetic programming), the trend toward decreasing prices of hardware will be even more important in terms of providing the large amounts of computational power necessary to solve realistic problems. In most genetic programming applications, the vast majority of computer resources are used on the fitness evaluations. The calculation of fitness for the individuals in the population is usually entirely decoupled. Thus, parallel computing techniques can be beneficially applied to genetic programming and genetic algorithms with almost 100% efficiency (Andre and Koza 1996). In fact, the use of semi-isolated subpopulations often accelerates the finding of a solution to a problem using genetic programming and produces superlinear speedup. Parallelization of genetic programming will be of central importance to the growth of the field.

H1.1.3.16 Evolvable hardware

One of the exciting new areas of evolutionary programming involves the use of evolvable hardware (Sanchez and Tomassini 1996). Evolvable hardware includes devices such as field programmable gate arrays (FPGAs) and field programmable analog arrays (FPAAs). These devices are reconfigurable with very short configuration times and download times. Thompson (1996) has pioneered the use of FPGAs to evolve a frequency discriminator circuit and a robot controller using the recently developed Xilinix 6216 chip. I anticipate an explosive growth in the use of genetic programming to evolve hardware and the use of reconfigurable hardware to accelerate genetic programming runs.

References

Andre D, Bennett F H III and Koza J R 1996 Discovery by genetic programming of a cellular automata rule that is better than any known rule for the majority classification problem *Genetic Programming 1996: Proc. 1st Ann. Conf. on Genetic Programming (Stanford, CA, July 1996)* ed J R Koza, D E Goldberg, D B Fogel and R L Riolo (Cambridge, MA: MIT Press)

Andre D and Koza J R 1996 Parallel genetic programming: a scalable implementation using the transputer network architecture *Advances in Genetic Programming 2* ed P J Angeline and K E Kinnear Jr (Cambridge, MA: MIT Press) ch 18

Bennett F H III 1996 Automatic creation of an efficient multi-agent architecture using genetic programming with architecture-altering operations *Genetic Programming 1996: Proc. 1st Ann. Conf. on Genetic Programming (Stanford, CA, July 1996)* ed J R Koza, D E Goldberg, D B Fogel and R L Riolo (Cambridge, MA: MIT Press)

Brave S 1995 Using genetic programming to evolve recursive programs for tree search *Proc. 4th Golden West Conf. on Intelligent Systems* (Raleigh, NC: International Society for Computers and Their Applications) pp 60–5

——1996a Evolving deterministic finite automata using cellular encoding *Genetic Programming 1996: Proc. 1st Ann. Conf. on Genetic Programming (Stanford, CA, July 1996)* ed J R Koza, D E Goldberg, D B Fogel and R L Riolo (Cambridge, MA: MIT Press)

——1996b The evolution of memory and mental models using genetic programming *Genetic Programming 1996: Proc. 1st Ann. Conf. on Genetic Programming (Stanford, CA, July 1996)* ed J R Koza, D E Goldberg, D B Fogel and R L Riolo (Cambridge, MA: MIT Press)

de Garis H 1996 CAM-BRAIN: the evolutionary engineering of a billion neuron artificial brain by 2001 which grows/evolves at electronic speeds inside a cellular automata machine (CAM) *Towards Evolvable Hardware* (*Lecture Notes in Computer Science 1062*) ed E Sanchez and M Tomassini (Berlin: Springer) pp 76–98

Floreano D and Mondada F 1994 Automatic creation of an autonomous agent: evolution of a neural-network drive robot *From Animals to Animats 3: Proc. 3rd Int. Conf. on Simulation of Adaptive Behavior* ed D Cliff, P Husbands, J-A Meyer and S W Wilson pp 421–30

Garces-Perez J, Schoenefeld D A and Wainwright R L 1996 Solving facility layout problems using genetic programming *Genetic Programming 1996: Proc. 1st Ann. Conf. on Genetic Programming (Stanford, CA, July 1996)* ed J R Koza, D E Goldberg, D B Fogel and R L Riolo (Cambridge, MA: MIT Press)

Gruau F 1994 Genetic micro programming of neural networks *Advances in Genetic Programming* ed K E Kinnear Jr (Cambridge, MA: The MIT Press) pp 495–518

Handley S 1996 A new class of function sets for solving sequence problems *Genetic Programming 1996: Proc. 1st Ann. Conf. on Genetic Programming (Stanford, CA, July 1996)* ed J R Koza, D E Goldberg, D B Fogel and R L Riolo (Cambridge, MA: MIT Press)

Holland J H 1975 *Adaptation in Natural and Artificial Systems: an Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence* (Ann Arbor, MI: University of Michigan Press) (1992 2nd edn, Cambridge, MA: MIT Press)

Howley B 1996 Genetic programming of near-minimum-time spacecraft attitude maneuvers *Genetic Programming 1996: Proc. 1st Ann. Conf. on Genetic Programming (Stanford, CA, July 1996)* ed J R Koza, D E Goldberg, D B Fogel and R L Riolo (Cambridge, MA: MIT Press)

Jacob C 1996 Evolving evolution programs: Genetic programming and L-systems *Genetic Programming 1996: Proc. 1st Ann. Conf. on Genetic Programming (Stanford, CA, July 1996)* ed J R Koza, D E Goldberg, D B Fogel and R L Riolo (Cambridge, MA: MIT Press)

Koza J R 1992 *Genetic Programming: on the Programming of Computers by Means of Natural Selection.* (Cambridge, MA: The MIT Press)

——1994 *Genetic Programming II: Automatic Discovery of Reusable Programs* (Cambridge, MA: MIT Press)

——1995 Gene duplication to enable genetic programming to concurrently evolve both the architecture and work-performing steps of a computer program *Proc. 14th Int. Joint Conf. on Artificial Intelligence* (San Mateo, CA: Morgan Kaufmann)

Koza J R and Andre D 1996a Classifying protein segments as transmembrane domains using architecture-altering operations in genetic programming *Advances in Genetic Programming II* ed P J Angeline and K E Kinnear Jr (Cambridge, MA: MIT Press)

——1996b Evolution of iteration in genetic programming *Evolutionary Programming V: Proc. 5th Ann. Conf. on Evolutionary Programming* (Cambridge, MA: MIT Press)

——1996c Automatic discovery of protein motifs using genetic programming *Evolutionary Computation: Theory and Applications* ed Yao Xin (Singapore: World Scientific)

Koza J R, Bennett III, Forrest H, Andre D and Keane M A 1996 Automated WYWIWYG design of both the topology and component values of analog electrical circuits using genetic programming *Genetic Programming 1996: Proc. 1st Ann. Conf. on Genetic Programming (Stanford, CA, July 1996)* ed J R Koza, D E Goldberg, D B Fogel and R L Riolo (Cambridge, MA: MIT Press)

Langdon W B 1996 Using data structures within genetic programming. *Genetic Programming 1996: Proc. 1st Ann. Conf. on Genetic Programming (Stanford, CA, July 1996)* ed J R Koza, D E Goldberg, D B Fogel and R L Riolo (Cambridge, MA: MIT Press)

Luke S and Spector L 1996 Evolving temwork and coordination with genetic programming *Genetic Programming 1996: Proc. 1st Ann. Conf. on Genetic Programming (Stanford, CA, July 1996)* ed J R Koza, D E Goldberg, D B Fogel and R L Riolo (Cambridge, MA: MIT Press)

Nordin P 1994 A compiling genetic programming system that directly manipulates the machine code *Advances in Genetic Programming* ed K E Kinnear Jr (Cambridge, MA: MIT Press)

Pollack J B and Blair A D 1996 Coevolution of a backgammon player *Artificial Life V: Proc. 5th Int. Workshop on the Synthesis and Simulation of Living Systems* (Cambridge, MA: MIT Press)

Rosca J P 1995 Genetic programming exploratory power and the discovery of functions *Evolutionary Programming IV: Proc. 4th Ann. Conf. on Evolutionary Programming (San Diego, CA, March 1995)* ed J R McDonnell, R G Reynolds and D B Fogel (Cambridge, MA: MIT Press)

Samuel A L 1959 Some studies in machine learning using the game of checkers *IBM J. Res. Dev.* **3** 210–29

Sanchez E and Tomassini M (eds) *Towards Evolvable Hardware* (*Lecture Notes in Computer Science 1062*) (Berlin: Springer)

Soule T, Foster J A and Dickinson J 1996 Code growth in genetic programming *Genetic Programming 1996: Proc. 1st Ann. Conf. on Genetic Programming (Stanford, CA, July 1996)* ed J R Koza, D E Goldberg, D B Fogel and R L Riolo (Cambridge, MA: MIT Press)

Spector L 1996 Simultaneous evolution of programs and their control structures *Advances in Genetic Programming 2* ed P J Angeline and K E Kinnear Jr (Cambridge, MA: MIT Press)

Teller A and Veloso M 1996 PADO: a new learning architecture for object recognition *Symbolic Visual Learning* ed K Ikeuchi and M Veloso (Oxford: Oxford University Press)

Thompson A 1996 Silicon evolution *Genetic Programming 1996: Proc. 1st Ann. Conf. on Genetic Programming (Stanford, CA, July 1996)* ed J R Koza, D E Goldberg, D B Fogel and R L Riolo (Cambridge, MA: MIT Press)

Walsh P and Ryan C 1996 Paragen: a novel technique for the autoparallelisation of sequential programs using genetic programming *Genetic Programming 1996: Proc. 1st Ann. Conf. on Genetic Programming (Stanford, CA, July 1996)* ed J R Koza, D E Goldberg, D B Fogel and R L Riolo (Cambridge, MA: MIT Press)

Whigham P A 1996 Search bias, language bias, and genetic programming *Genetic Programming 1996: Proc. 1st Ann. Conf. on Genetic Programming (Stanford, CA, July 1996)* ed J R Koza, D E Goldberg, D B Fogel and R L Riolo (Cambridge, MA: MIT Press)

H1.2 Future research in evolutionary computation

Lawrence J Fogel

Abstract

Future directions for research in evolutionary computation are considered and offered. Specific attention is devoted to making progress in mathematical theory, empirical assessments, self-adaptation, and the understanding of coevolution and self-organizing systems, particularly as they pertain to natural evolution.

Despite significant improvements in our understanding of the fundamentals of evolutionary computation, there are many open problems and directions for future research. Some are presented here, but there are also many other important areas. The directions we should take are a function of our purpose in using evolutionary computation. For many, the utility of evolutionary computation lies in solving engineering problems; for others, simulated evolution is a tool for gaining a better understanding of the potential of natural evolution, or for learning about the behavior of competing entities in arbitrary settings. There are several possibilities to explore in each of these regards.

Practical problem solving requires reliable and rapid optimization. Evolutionary algorithms have demonstrated an ability to quickly discover useful solutions to problems that have been difficult to solve using classical optimization techniques (see e.g. Gehlhaar *et al* 1995), but there has been a shortage of theoretical framework on which to build an understanding of the effects of various parametrizations within an evolutionary algorithm. Most of the practical results have dealt with convergence rates on strongly convex problems, or problems that can be approximated as being convex (Bäck 1996), but these are not the problems of real interest. Moreover, analysis of the expected number of substrings or so-called building blocks in arbitrary problems has been generally useless in devising settings or conditions for improving the performance of evolutionary algorithms. Such piecemeal analysis appears most unsuitable for addressing complex problems with interacting components. Mathematical analysis is likely to depend on specific characteristics of each function studied, and will therefore be brittle rather than general. Although the hope of improving evolutionary algorithms in general function optimization on the basis of mathematical theory appears dim, such results would be most valuable.

For want of such theory, however, empirical investigations and comparisons between different algorithms in different functional settings must continue. Such comparisons have been useful in identifying the suitability of particular operators, or their unsuitability, since at least the work of Reed *et al* (1967). Recent efforts by Altenberg (1995), Grefenstette (1995), Fogel and Ghozeil (1996), and Voigt *et al* (1996) all are giving more attention to understanding the phenotypic effect of random variation in light of a selection function. Empirical investigation on a series of functions in a systematic fashion may yield insight into the optimal parametrizations for certain classes of functions. This would be useful and practical information.

Another clearly important avenue for further investigation is the use of *self-adaptation*, allowing an evolutionary search to adapt the manner in which it distributes trials in light of information gleaned during the optimization process. These efforts have a long history, again going back at least to the work of Reed *et al* (1967), but have more recently focused on parameter optimization in determining the appropriate mutation variance in evolutionary algorithms (Bäck and Schwefel 1993), or the crossover locations or forms in genetic algorithms (Spears 1995). Davis (1994) offered a first look at self-adaptation of the general form of the mutation distribution (i.e. it need not follow a known parametric density function such as a Gaussian). This work has received less attention than it deserves. It should be possible to engineer

C7.1

methods for introducing self-adaptation of arbitrary random variation functions in light of multiple solutions in the population as well as information gleaned from past generations.

The solution of complex engineering problems is important, but the use of evolutionary algorithms need not be restricted to mere function optimization. The methods can also be used to gain an understanding of how competitive or cooperative agents may interact given a variety of different available resources and purposes. These circumstances present complex coevolutionary systems in which multiple agents adapt their behavior to meet goals in time varying environments. It appears that *simulated evolution* may be of F1.10 benefit to understanding the manner in which such agents might interact, but many of the attempts to use evolutionary computation in these cases have been misdirected. The common use of emulating specific genetic mechanisms found to operate on deoxyribonucleic acid (DNA) within individuals in simulations that are then applied to agents at higher levels in the hierarchy of evolutionary units is inappropriate. For example, there is no reason to apply methods akin to one-point crossover when simulating the interaction of competing agents in economic systems, such as corporations (cf Holland 1995, pp 84–7). One-point crossover does not occur at this level of abstraction, and even if it did, it is not at all clear that its effect on the behavior of the agents in the simulation would be analogous to the effect of creating new companies by cutting apart and splicing together existing ones! A greater understanding that the trajectory an evolutionary system takes is primarily dependent on the adaptive landscape that measures behavior (often in terms of behavioral error) and the effects of the random variation operators that can modify that behavior would represent a major breakthrough.

The area of self-organizing systems has long been of interest to practitioners of evolutionary computation and many problems remain in this area as well. Foremost is the problem of gaining an understanding of *metamerism*, the process in which a unit structure is duplicated a number of times and during this process is reoptimized for other uses (see the article by Atmar (1994) for a complete discussion). There have been some recent interesting efforts to explicitly use duplicated code in genetic programming (see e.g. Koza 1995). But to gain a real understanding of the process of metamerism such duplication cannot be forced by the human operator. The conditions for inventing the process must be given, rather than the process itself. It is fair to say that these conditions are unknown presently.

Perhaps the greatest challenge facing evolutionary computation is its use as a means for gaining a greater understanding of natural evolution. This has been the promise of the efforts of artificial life, but like many other such promises throughout the course of computer science, they have been left mainly unfulfilled. Simulations can be of use to address ecological systems in which mathematical descriptions are either not possible or unwieldy (see e.g. Fogel and Fogel 1995), but even in these cases it is important to recognize the elements of the natural system that are omitted from the models and attempt to understand the effect that such omission may have on the results. Moreover, it is critical to recognize that the behavior of complex adaptive systems, by definition, relies on the interactions of many components and that the measurable success of such complex systems can never be decomposed and attributed to any of the individual components. An important step forward could be realized if attempts to perform such credit assignment and related schema analysis in complex systems were abandoned in favor of more holistic understandings of how selection acts on complex sets of behaviors in concert, rather than in isolation. Just as no general understanding of the physics of flight can come from assigning credit to feathers or flapping wings, no general understanding of complex adaptive systems can come from piecemeal analysis of their 'genes'. Refocusing attention on 'organisms' rather than 'genes' represents a compelling and promising, although old, direction for further investigation.

References

Altenberg L 1995 The schema theorem and Price's theorem *Foundations of Genetic Algorithms* vol 3, ed L D Whitley and M D Vose (San Mateo, CA: Morgan Kaufmann) pp 23–49

Atmar W 1994 Notes on the simulation of evolution *IEEE Trans. Neural Networks* **NN-5** 130–47

Bäck T 1996 *Evolutionary Algorithms in Theory and Practice* (New York: Oxford University Press)

Bäck T and Schwefel H-P 1993 An overview of evolutionary algorithms for parameter optimization *Evolutionary Comput.* **1** 1–24

Davis M W 1994 The natural formation of Gaussian mutation strategies in evolutionary programming *Proc. 3rd Ann. Conf. on Evolutionary Programming (San Diego, CA, February 1994)* ed A V Sebald and L J Fogel (Singapore: World Scientific) pp 242–52

Fogel D B and Fogel G B 1995 Evolutionary stable strategies are not always stable under evolutionary dynamics *Evolutionary Programming IV: Proc. 4th Ann. Conf. on Evolutionary Programming (San Diego, CA, March 1995)* ed J R McDonnell, R G Reynolds and D B Fogel (Cambridge, MA: MIT Press) pp 565–77

Fogel D B and Ghozeil A 1996 Using fitness distributions to design more efficient evolutionary computations *Proc. 1996 IEEE Conf. on Evolutionary Computation* (Piscataway, NJ: IEEE) pp 11–9

Gehlhaar D K, Verkhivker G M, Rejto P A, Sherman C J, Fogel D B, Fogel L J and Freer S T 1995 Molecular recognition of the inhibitor AG-1343 by HIV-1 protease: conformationally flexible docking by evolutionary programming *Chem. Biol.* **2** 317–24

Grefenstette J J 1995 Predictive models using fitness distributions of genetic operators *Foundations of Genetic Algorithms* vol 3, ed L D Whitley and M D Vose (San Mateo, CA: Morgan Kaufmann) pp 139–61

Holland J H 1995 *Hidden Order: How Adaptation Builds Complexity* (Reading, MA: Addison-Wesley)

Koza J R 1995 Evolving the architecture of a multi-part program in genetic programming using architecture-altering operations *Evolutionary Programming IV: Proc. 4th Ann. Conf. on Evolutionary Programming (San Diego, CA, March 1995)* ed J R McDonnell, R G Reynolds and D B Fogel (Cambridge, MA: MIT Press) pp 695–717

Reed J, Toombs R and Barricelli N A 1967 Simulation of biological evolution and machine learning *J. Theor. Biol.* **17** 319–42

Spears W M 1995 Adapting crossover in evolutionary algorithms *Evolutionary Programming IV: Proc. 4th Ann. Conf. on Evolutionary Programming (San Diego, CA, March 1995)* ed J R McDonnell, R G Reynolds and D B Fogel (Cambridge, MA: MIT Press) pp 367–84

Voigt H-M, Mühlenbein H and Schlierkamp-Voosen D 1996 The response to selection equation for skew fitness distributions *Proc. 1996 IEEE Conf. on Evolutionary Computation* (New York: IEEE) pp 820–5

H1.3 Challenges to and future developments of evolutionary algorithms

Hans-Paul Schwefel

Abstract

Any research field with many open questions and controversial beliefs has a potentially prosperous future, and that of evolutionary algorithms is no exception. There are many gaps in the theoretical background of the algorithms already in use, and there is a bulk of as yet unincorporated phenomena and mechanisms of organic evolution underpinning the hope for further breakthroughs in devising ever more useful evolutionary algorithms. Some of them are outlined in this section.

Since evolutionary algorithms (EAs) try to mimic an important aspect of real life, it is worthwhile to look at the roots before striving to add leaves to the tree. Thus, let me begin with some controversial beliefs about the nature of the evolutionary process. Such convictions have hindered and still jeopardize the modeling substantially. The oldest model of evolution, still believed in some circles, simply denies its existence. This will not be debated here.

If consulting some older encyclopedia, one still finds under 'evolution' an explanation such as 'gradual development from simpler to higher forms of life', which is in sharp contrast to the observation of so-called punctuated equilibria that Eldredge and Gould (1972) think they have found in fossil remnants of living beings. van Nimwegen *et al* (1996) recently proved that such episodes happen in finite *genetic algorithm* B1.2 (GA) populations as well, and Schwefel (1987) has reported the same for *evolution strategies* (ESs) in cases B1.3 of too strong selection pressure, such that the individuals try to gain quick success in lower-dimensional subspaces.

Another model has been that of a pure random search, sometimes called the trial-and-error or Monte Carlo method (Ashby 1960). This model has been (mis-) used to demonstrate the improbability of assembling even a simple wristwatch in ten billion years, let alone living beings—in order to deny evolution at all.

Current evolutionary algorithms are certainly better models of organic evolution. Nevertheless, they are still far from being isomorphic mappings of what happens in nature. In order to perform better, an appropriate model of evolution would have to comprise the full temporal and spatial development on the earth (a real global model) if not within the whole universe. We must be more modest in order to understand at least a little of what really happens—as always within the natural sciences.

Let us look first at some deficiencies of current EAs compared to organic evolution. The benefit will be that this kind of approach may lead directly to further improvements in evolutionary computation, either by increasing the efficiency or the effectivity and/or range of applicability of EAs.

Organic evolution certainly does not only aim at finding static optima just once and with ultimate precision. Organic evolution happens within an ever-changing environment, where evolvability is more important than precision. Only subsystems such as blood vessels, where long-term constancy of physical laws of hydrodynamics prevails, may adapt in the sense of optimizing diameter ratios at branchings of the system according to a minimal-effort criterion for pumping the blood and maintaining the vessels themselves (Cohn 1954). In general, the environment is not only intrinsically dynamic; it is changed by the mutual actions of all participants in the evolutionary game. From the perspective of one species, the search for meliorization (a term preferred to optimization by the author) takes place on a trampoline,

deformed by (re-)actions of other species, as well. A multispecies EA thus should be applicable to dynamic optimum-holding tasks, for example. Nature demonstrates that even catastrophic events with great losses of whole species can always be overcome. Individuals' and even species' finite life spans might be a necessary ingredient for the system's survivability (Schwefel 1987, Kursawe 1992). However individual adaptivity to new conditions could be another means of approaching optimum-holding tasks. The human immune system is a good example here. Nobody doubts that its adaptation ability is also encoded as a program. The time scale of responses is much shorter, however, than a generational cycle.

The next important fact is that organic evolution always deals with a situation of multiple selection criteria. These may be hard constraints as given by physical laws that must be met; otherwise lethal trials result. However, even if that is taken into account, there are in most cases different predators that test a new individual according to several single criteria during their life span. Since there is no general selector who averages or weighs the risks and performances, the evolutionary meliorization can never end up with a single solution even under stationary environmental conditions. There are always many simultaneously equally good solutions (the so-called *Pareto set* of nondominated or efficient solutions). This may be one C4.5.3 reason why there are so many species, not even half precisely counted so far. Again, this observation may be turned around by stating that a proper EA should be able to solve vector optimization problems, even resulting in a whole set of Pareto-optimal solutions during one run. This has been demonstrated already by using diploid individuals and simulating dominance–recessivity (Kursawe 1991). However there may also be other mechanisms that help in devising even better *multiple-criterion decision-making* (MCDM) EAs. F1.9 Two overview articles may be of value for the interested reader, one by Fonseca and Fleming (1995), the other by Tamaki *et al* (1996).

Most modern living beings are multicellular, each cell comprising the full genetic information of its individual. Not only during adolescence, but also during the whole life span of an individual, the cells divide, copy their information content and thus incur underlying errors during that process (somatic mutations). Any phenotypic feature depending on an ensemble of cells thus is not in direct correspondence with the genotype but somehow fuzzy—according to the mutability which may be encoded genetically, as well. Assuming similarity between somatic and genetic mutabilities and a selection process evaluating ensemble averages of cell tissues, it was demonstrated that even binary problems could be solved by means of an ES with self-adaptive individual mutation rates (Schwefel 1975). No further investigations have been performed to date to make use of such phenomena in EAs.

The correspondence between genotype and phenotype may be even more complicated through polygeny and pleiotropy, or more generally, through what is called the epigenetic apparatus. The latter is not fixed but encoded—somehow—in the genome (including mitochondria) itself. The genetic code had to be learned during evolution, too. Though it is fixed today and nearly the same globally, we should remember that we are normally beginning from scratch in artificial evolution. A similar argument, by the way, holds for the proper mutation rate for discrete variables. Very low rates as observed in temporary optimal or near-equilibrium situations may not be good for starting conditions.

It is an intriguing question whether introns ('noncoding' sections of the genome) might be involved in such internal regulating processes that are simply not observable when looking at the phenotype only. ESs rely on such internal parameters (as opposed to object parameters) encoded in the individuals' genomes for on-line self-adaptation of variances and covariances of phenotypic mutations. The real-world case may be even more flexible in allowing adaptation to much more complicated genotype–phenotype mappings and thus creation of internal models of the individuals' environment, or, in other words, variation patterns that take into account the natural laws, especially those which must be closely met for survival.

Autoadaptation of strategy parameters, be they discrete or real valued, can and has been handled in two different ways: these parameters vary either from individual to individual or from subgroup to subgroup. The latter concept, sometimes called metaevolution, sometimes nested strategy, has been reported to perform well in structural optimization with ESs (Lohmann 1992) where the discrete object parameters were handled within the outer loop, but the natural counterpart is at least not clear. Kursawe (1995) has used a metaevolutionary concept successfully to learn the proper recombination type and the number of different variances within a multimembered ES, but there is no reason why this could not be done on-line within ESs in the same manner as with variances and covariances. The possibilities of on-line adaptation of a variable epigenetic apparatus are by far not exhausted in all EAs today.

Although most implementations of EAs use an operator called recombination or crossover (except for evolutionary programming (EP) versions), thus resembling the mix of parental chromosome parts in composing a newborn descendant, they do not yet handle two (or more) sexes really. The simulated

treatment of two sexes (Miller and Todd 1993) has shown interesting patterns of behavior, but this has not led to a purposeful employment in optimization.

One should not force political correctness, good in the human sphere, when looking at more 'primitive' species such as bacteria and yeasts. It may be worthwhile to ask whether in some cases the sexes underlie different selection criteria, at least in mating selection, which is too often confused with environmental selection. This may yield a fresh view on the case of MCDM, either by taking into account different constraints or objective functions in evaluating male and female descendants to become parents of the next generation. Constrained optimization problems as well as vector optimization problems are especially difficult insofar as further improvements are restricted to a sometimes tiny cone within the set of neighboring solutions. Splitting up the criteria among sexes might help in ridge riding constraints or Pareto-optimal sets.

Larger populations in the real world are spatially distributed (small populations are always prone to extinction). Neither mating nor predation thus takes place in a way that includes all individuals with equal chance. There is no global selection in either case. Whereas *tournament selection* may be a good model C2.3
of predation when applied to neighbors only, mating selection in such a neighborhood model is still open to different forms of modeling. Either differences among mates are attractive or similarities act in such a way (see Goethe and other poets), or, as scientists working with guinea-pigs report, average behavior is most attractive in choosing mates. No attempts to look into the modeling of such hypotheses and their consequences for evolutionary computation are known, so far. Within spatially organized EAs migration and diffusion principles have already done a good job for solving multimodal optimization tasks, i.e. a field in which a proper balance between the conflicting goals of high (parallel) efficiency and effectiveness has to be found. Consequently modeling selection as an asynchronous and spatially distributed process with several predators at a time among the prey, which is the normal case in ecological systems, has not yet been done but should be considered.

The genome of an individual can be looked at as a program not for a fixed outcome, but for a process yielding a product depending on the actual environment a newborn descendant is confronted with. Boseniuk and Ebeling (1991) have given a pointer in this direction by devising a combined Darwin–Haeckel model of evolution, the latter part resembling the ontogeny of an individual. This is simulated here by a greedy local search starting from the inherited position of an individual. More generally, a Haeckel component of an evolutionary algorithm can be taken as any kind of individual adaptation to the local–temporal environmental conditions; but the result of this cannot be transferred genetically to the next generation. Completely different from this mechanism is the life-long individual learning and all kinds of social transmission of the knowledge (and prejudice) gained this way. EA individuals so far have no kind of brain of their own but this might be added in the future.

Many attacks against EAs as optimization, or, better, meliorization, algorithms have been mounted by those who emphasize cooperative behavior within evolutionary processes. This controversy is not necessary. Cooperative problem solving by sharing resources or dividing the problem into subproblems to conquer may lead to novel approaches not yet taken into account thoroughly. At least, theoretical results are missing. A reference to *classifier systems* (CSs) and multiagent systems may be appropriate and must B1.5.2
be sufficient here.

Whereas the inversion of parts of the genome has been tried, though with minor successes only, the idea of changing the genome length during simulated evolution has not been considered to the extent it deserves. Gene duplication as well as gene deletion, however, really take place in organic evolution. Modeling this phenomenon has had much success in solving a nozzle shaping problem experimentally (Klockgether and Schwefel 1970)—with hot water from a steam generator instead of a computer. Genome length variation should be an important ingredient in all cases of so-called structure optimization tasks, where the number of variables is not known in advance, but is a variable itself. Redundancy in the genotype–phenotype mapping is a matter of fact in organic evolution. It obviously helps to smooth the *fitness landscape*. Whether it also helps in finding extradimensional bypasses in difficult optimization B2.7
tasks, as suggested by Conrad (1993), is an interesting question not yet satisfactorily answered.

If it were for the sake of devising numerical optimization methods only, the idea of making use of natural metaphors would have a weak background. There will always exist nonnatural algorithms to solve specialized classes of optimization problems. The suggestion of an ever-existing 'remainder of problems', which EAs may be good for, is a weak argument for their existence.

A more important argument for investigating EAs is the fact that they model mechanisms found not only in biology but in a more or less similar way in other systems. The nearest field is ecology, where

evolutionary principles certainly are at work, but even economy may be looked at as a field where natural forces are at work. Human actions are perhaps still more natural than artificial, even if human pride tends to rankle at such a remark.

There has been a tendency to understand ecology and economy as adaptation and equilibration processes only, but that is less than half of the truth. Both fields deal in their long-term versions with phenomena that are generally perceived as either steady improvement (progress) or steady deterioration.

Many more ideas for improving EAs may come forth when biologists and computer scientists or engineers/managers using EAs in design, control, and management, or other decision-making processes, sit together without prejudice concerning the scholarliness of their points of view. Obstacles to further success do not lie in the real world; they only lie in the heads of those who try to stay self-contained within their narrow single disciplines. The field of 'directions for future research', in principle, is wide open and—certainly—full of chance and surprises.

References

Ashby W R 1960 *Design for a Brain* 2nd edn (New York: Wiley)

Boseniuk T and Ebeling W 1991 Boltzmann-, Darwin-, and Haeckel-strategies in optimization problems *Parallel Problem Solving from Nature—Proc. 1st Workshop PPSN I (Lecture Notes in Computer Science 496)* ed H-P Schwefel and R Männer (Berlin: Springer) pp 430–44

Cohn D L 1954 Optimal systems I—the vascular system *Bull. Math. Biophys.* **16** 59–74

Conrad M 1993 Structuring adaptive surfaces for effective evolution *Proc. 2nd Ann. Conf. on Evolutionary Programming (San Diego, CA, 1993)* ed D B Fogel and W Atmar (La Jolla, CA: Evolutionary Programming Society) pp 1–10

Eldredge N and Gould S J 1972 Punctuated equilibria: an alternative to phyletic gradualism *Models in Paleobiology* ed M J Schopf (San Francisco, CA: Greeman and Cooper) pp 82–115

Fonseca C M and Fleming P J 1995 An overview of evolutionary algorithms in multiobjective optimization *Evolutionary Comput.* **3** 1–16

Klockgether J and Schwefel H-P 1970 Two-phase nozzle and hollow core jet experiments *Proc. 11 Symp. on Engineering Aspects of Magnetohydrodynamics* ed D G Elliott (Pasadena, CA: California Institute of Technology) pp 141–8

Kursawe F 1991 A variant of evolution strategies for vector optimization *Parallel Problem Solving from Nature—Proc. 1st Workshop PPSN I (Lecture Notes in Computer Science 496)* ed H-P Schwefel and R Männer (Berlin: Springer) pp 193–7

——1992 Naturanaloge Optimierverfahren—Neuere Entwicklungen in der Informatik *Studien zur Evolutorischen Ökonomik II, Schriften des Vereins für Socialpolitik* vol 195 II, ed U Witt (Berlin: Duncker and Humblot) pp 11–38

——1995 Towards self-adapting evolution strategies *Proc. 1995 IEEE Conf. on Evolutionary Computation* ed Y Attikiouzel and C deSilva (Piscataway, NJ: IEEE) pp 283–8

Lohmann R 1992 Structure evolution and incomplete induction *Parallel Problem Solving from Nature 2* ed R Männer and B Manderick (Amsterdam: Elsevier) pp 175–85

Miller G F and Todd P M 1993 Evolutionary wanderlust: sexual selection with directional mate preferences *From Animals to Animats 2, Proc. 2nd Int. Conf. Simulation of Adaptive Behavior* ed J-A Meyer, H L Roitblat and S W Wilson (Cambridge, MA: MIT Press) pp 21–30

Schwefel H-P 1975 *Binäre Optimierung durch somatische Mutation* Technical Report, Working Group of Bionics and Evolution Techniques at the Institute of Measurement and Control Technology of the Technical University of Berlin and of the Central Animal Laboratory of the Medical High School of Hanover

——1987 Collective phenomena in evolutionary systems Problems of constancy and change—the complementarity of systems approaches to complexity *31st Ann. Meeting Int. Soc. Gen. Syst. Res. (Budapest)* vol 2, ed P Checkland and I Kiss (Int. Soc. for General System Research) pp 1025–33

Tamaki H, Kita H and Kobayashi S 1996 Multi-objective optimization by genetic algorithms: a review *Proc. 1996 IEEE Conf. on Evolutionary Computation* ed T Fukuda, T Furuhashi and D B Fogel (Piscataway, NJ: IEEE) pp 517–22

van Nimwegen E, Crutchfield J P and Mitchell M 1996 *Finite Populations induce Metastability in Evolutionary Search* Santa Fe Institute Working Paper 96-08-054