To Decode Short Cryptograms

George W. Hart

Short cryptograms, in which an encoded sentence or quotation is to be decoded, are common pastimes of many recreational puzzle enthusiasts. Here is a simple example of the type that can be found in many collections of word games, and daily in some newspapers:

Given (cipher text): YPNR,PT MPYYPNR:YJSYODYJR WIRDYOPM.
Solution (plain text): TOBE,OR NOTTOBE:THAT IS THEQUESTION.

A permutation of the 26-character alphabet is used to encode a sentence with spacing and punctuation intact. Given only the encoded sentence (the "cipher text"), the correct permutation is to be found so that the original sentence (the "plain text") can be understood. By framing the problem as a multiple-hypothesis detection problem, applying a maximum-likelihood criterion, using English language word frequency data, approximating liberally, and constructing a well-organized search tree, a rather simple algorithm results, which quickly deciphers even difficult cryptograms.

Cryptograms of this form—simple permutation substitutions with word divisions—have been employed for message concealment, at least, since Roman times. The solution of simple permutation ciphers has not been of much practical importance, since their use for military communication was superseded in the nineteenth century, but they remain a formidable puzzle for those who enjoy word games. Experienced solvers can manually solve a typical one-sentence cryptogram in a few minutes, but carefully constructed short puzzles, with unusual letter frequencies or atypical letter combinations, can stymie even expert solvers. Many strategies are published for manual decipherment, e.g., [1–3, 5, 8–10, 12], but these all require human pattern recognition skills "in the loop," and are not explicit enough to be called algorithms. This author is aware of only one previously published method for automatic solution—a relaxation method [7], also see [4]—but it is not suitable for short cryptograms.

The solution of a cryptogram can either be given as an explicit sentence of plain text, or it can be characterized by describing the permutation that was used to code the plain text. This permutation can be inverted then to reconstruct the plain text from the given cipher text. The permutation used for this example codes each letter as the letter to its right on the standard typewriter keyboard (convenient for touch-typists), with the three letters on the extreme right, (P, L, and M) "wrapping around" to the left:

Plain:	ABCDEFGHIJKLMNOPQRSTUVWXYZ
Cipher:	SNVFRGHJOKLAZMPQWTDYIBECUX
Partial Permutation:	SN**R**JO****MP*WTDYI*****

An algorithm that decodes cryptograms must effectively choose one of the 26! ($\approx 4 \cdot 10^{26}$) different permutations according to some criterion. Actually, there are usually somewhat fewer possibilities, because only a "partial permutation" is required. Only 12 distinct letters appear in the preceding quote, so cipher entries for the plain letters that do not appear may be left undefined, indicated here with an (asterisk) *.

Criteria

The first tool one might think of when constructing a selection criterion is a probability distribution for the 26 letters. By tabulating the occurrences of each alphabetic character in large samples of text, one determines that the most frequent letter is E, occurring about 13% of the time, while the least common is Z, with a frequency about 0.1%. Complete rank orderings vary somewhat depending on the body of the text selected for tabulation. Four published examples for modern English [2, 3, 9, 12] are:

ETAONISRHLDCUPFMWYBGVKQXJZ
ETNRIOASDHLCFPUMYGWVBXKQJZ
ETAOINSRHLDCUMFWGYPBVKXJQZ
ETOANIRSHDLUCMPFYWGBVKJXZQ

A natural solution criterion which can make use of a

given ordering is to select a permutation that results in a hypothesized plain text with as close an ordering as possible to a standard published ordering. Variations on this idea appear in all discussions of cryptography. The existence of conflicting nominal orderings, such as the preceding four is only a minor flaw with this approach; there is a much more serious problem. While it is straightforward to construct a permutation that results in the desired letter ordering, the result of such an algorithm (the "output text") will, almost certainly, be gibberish for a short length of text. Figure 1 illustrates such an algorithm, and why it fails. The fundamental problem is that short text fragments have sample statistics that differ considerably from one another and larger samples. Even large samples vary, as the four orderings above attest, so the use of letter frequencies alone may fail even with texts as large as 10,000 letters.

Similar arguments and examples can be constructed to show that simple modifications of this character-based probability distribution criterion also do not work in short cryptograms. Putative methods might include the frequencies of word-initial letters or word-final letters, or the joint statistics of pairs or triples of letters, using Markov models such as in [5, 8, 10]. However these considerations only exacerbate the problems, since only a very small fraction of the possible n-tuples appears. The fundamental problem remains that there is a large variance to the sample statistics of short segments of text. These measures can only be expected to converge when large samples of cipher text are available. The method of [7] uses letter triples, but based on the examples presented, it apparently requires approximately 1,000 characters of text.

To solve typical cryptograms containing only 5 to 25 words, a different tack is employed here, analogous to the method of Figure 1, but using complete words rather than letters. A word-based approach appears formidable at first because there are many more words in English than letters—over 100,000. However, it turns out that the use of a word table on the order of 100 to 1,000 entries allows for a very effective method of solution.

As an appropriate criterion, a maximum-likelihood (ML) estimator is used, justified by the fact that it gives a minimum probability of error under the assumption that all permutations are equally likely [11]. Let f represent an encoding permutation of the alphabet, applied to the plain-text string on a character-by-character basis. Let Z be the given cipher text. The corresponding plain text is then $f^{-1}(Z)$. A probabilistic model for natural language text assigns a probability $P(S)$ to any string S. The ML criterion is then to chose the permutation

$$f = \underset{f}{\operatorname{argmax}} P(f^{-1}(Z)). \tag{1}$$

There are two sizable problems: determining an appropriate probability distribution P for English sentences and choosing among the 26! values for the argument f.

Language Model

It is not clear that the notion of a probability distribution for English sentences makes any mathematical, linguistic, or philosophical sense. People do not decide what to say or write by any procedure analogous to flipping coins. It is more correct to describe the following as a text model based on word frequencies.

Many tabulations of word frequencies have been undertaken. Here is one listing of the 135 top-ranked words of modern American English, starting with the most common [6]:

THE OF AND TO A IN THAT IS WAS HE FOR IT WITH AS HIS ON BE AT BY I THIS HAD NOT ARE BUT FROM OR HAVE AN THEY WHICH ONE YOU WERE HER ALL SHE THERE WOULD THEIR WE HIM BEEN HAS WHEN WHO WILL MORE NO IF OUT SO SAID WHAT UP ITS ABOUT INTO THAN THEM CAN ONLY OTHER NEW SOME COULD TIME THESE TWO MAY THEN DO FIRST ANY MY NOW SUCH LIKE OUR OVER MAN ME EVEN MOST MADE AFTER ALSO DID MANY BEFORE MUST THROUGH BACK YEARS WHERE MUCH YOUR WAY WELL DOWN SHOULD BECAUSE EACH JUST THOSE PEOPLE MR HOW TOO LIT-

Figure 1. A possible "message" which matches the known letter frequencies of English can always be found by counting the occurrences of each cipher text letter and ranking them from most to least frequent (1st two columns), then matching with a known ordering (ETAONISRHLDC . . . in 3rd column). The output of this algorithm comes as close as possible to the correct letter frequencies of English, but is gibberish. The problem is that typical sentences of English do not display the actual statistics of English because they are too short. In the quoted line of Shakespeare, for example, "T" is the most common letter (column 4), not "E."

Given Cipher Text: YP NR, PT MPY YP NR: YJSY OD YJR WIRDYOPM.

Cipher chars.	Number of occurences	Plain text chars. in freq. order	Actual plain text char.
Y	7	E	T
P	5	T	O
R	4	A	E
N,M,J,O,D	2	O,N,I,S,R	B,N,H,I,S
T,S,W,I	1	H,L,D,C	R,A,Q,U

Resulting Output Text ET OA, TH NTE ET OA: EILE SR EIA DCARESTN.

TLE STATE GOOD VERY MAKE WORLD STILL OWN SEE MEN WORK LONG GET HERE BETWEEN BOTH LIFE BEING UNDER NEVER DAY SAME ANOTHER KNOW WHILE LAST (2)

Surprisingly, although there are over 100,000 English words, a randomly selected word of an English sentence has a greater than 50% chance of being found in this list. In tabulations of this sort, homographs (e.g., the word *that* that introduces relative clauses and the identically spelled pronoun *that*) are counted together, so the data is ideal for this text-based purpose (and arguably of little other use).

An interesting property of this data is that when the frequency of each word is plotted vs. its rank on logarithmic axes, a nearly linear relationship results [6]. The slope is very close to -1, and a good approximation to this data is the line

$$p(w) \approx 0.07/R(w), \tag{3}$$

where $R(w)$ is the rank and $p(w)$ the probability (or frequency) of the word w. For example, the most common word, THE, has rank 1 and appears as 7% of all words written. The second ranked word, OF, has half this frequency; it occurs as 3.6% of all written words. Frequency continues to decrease with rank in a nearly reciprocal manner. (Of course it stops obeying (3) at some point, since a harmonic series diverges yet the total must be unity.)

Therefore, a very short list of words constitutes a sizable fraction of any typical body of text. The first two already total to over 10% of all written words. By summing (3), one determines that the top 135 words form half of all printed text, leaving the remaining 100,000 English words to make up the second half. Thus, almost all English words have a probability less than 10^{-6} and so can be ignored as extremely unlikely to appear in any given sentence.

The trick to an effective deciphering algorithm is to find a rough approximation, simplifying (3), which will allow a rapid construction of the most likely permutation, rather than a search through the 26! possible permutations. The approximation used here is this:

If a word is in (2), it has a probability on the order of 10^{-2}, and if it is not in (2), it has a probability on the order of 10^{-6}. (4)

Although this is a rather coarse approximation, its justification is that the resulting algorithm works.

The final modeling step is to construct a probability measure for the sentences of English, as a function of the words in the sentence. Again, a very simple approximation is found to be sufficient. The simplest model one might construct is that the words in a sentence are generated completely independently of one another, so the probability of the entire sentence is simply the product of the probabilities of the individual words. More formally, for an N-word sentence,

$$P(S) = \prod_{i=1}^{N} p(w_i), \tag{5}$$

where the sentence S consists of words $w_1 \ldots w_N$. This is totally nonlinguistic of course, since it is independent of the most fundamental syntactic principles such as word order. But again the justification is in the results.

We can interpret (5) in two ways, according to how we count repeated words. When a given word appears k times in a single sentence, we could include the corresponding probability k times in the product (if we are counting word "occurrences") or include it only once (counting word "types"). The decision is immaterial for most sentences, but counting types is preferable, in order to avoid being misled in those sentences where an uncommon word type has several occurrences (e.g., *Romeo, Romeo, wherefore art thou Romeo?*) So our model depends only on the set of words found in the sentence, ignoring order and repetitions.

Because the probabilities (4) combine multiplicatively in (5), the criterion (1) is equivalent to:

Choose the permutation which makes as many word (types) as possible of the resulting output text be in the dictionary (2). (6)

In hindsight this is a natural criterion without any recourse to ML estimation and particular probabilistic models, but it is insightful to understand it in terms quitexioof more general principles, especially when considering variations on the method.

Optimization

Rather than search through the very large space of up to 26! partial permutations, the algorithm searches through a much more manageable tree of word assignments. First, for each cipher-text word, consider the relatively small set of plain-text words from (2) which could result from a deciphering permutation. Call this the *pattern set*, $s(w)$, for the word w. Such words must be of the same length and have the same pattern of repeated letters if any repeat. Thus, THAT, with its first and fourth letters equal, would not match WITH, but it does match HIGH, DEAD, and SAYS. Figure 2 lists the pattern sets for the coded words in the example. Note that there are generally fewer such sets than word types in the cipher, because several cipher words will share the same pattern. In this example, the eight cipher word types result in four pattern sets. The last is the empty set because there are no eight-letter words in (2). QUESTION is ranked 358th in [6], and so is not in the top 135.

For each cipher-text word w, $s(w)$ can be constructed by reading the dictionary a word at a time and comparing the plain-text words to the cipher words for length and repetition positions.[1] This can be accomplished with the following function of the two words:

```
FUNCTION MATCH?(W1, W2) {
  IF Length(W1) ≠ Length(W2) THEN RETURN FALSE;
  FOR I = 2 TO Length(W1) DO
    FOR J = 1 TO I - 1 DO
      IF (W1[I] = W1[J]) ≠ (W2[I] = W2[J]) THEN RETURN FALSE;
  RETURN TRUE }
```

The test in the inner loop causes the function to report

[1]The dictionary can be organized into patterns ahead of time to save some execution time, but this is not a dominant factor in the overall time complexity of the algorithm.

that the two words do not match when character positions i and j can be found: such that in one word the ith and jth positions contain the same character, but in the other word they contain different characters. If no such i and j can be found, the function reports TRUE in the last line. Then, $s(w)$ is the set of words x from the dictionary such that MATCH? (w, x) returns TRUE.

The full search tree is indicated in Figure 3. Each node is associated with a partial permutation, a set of cipher-word-to-plain-word assignments, and a score. The branching at level i represents all the plain-text words of which cipher word w_i could be an encoding i.e., $s(w_i)$, plus one further possibility (the rightmost in each set of siblings) that the cipher-text word might be an encoding of a word not in the dictionary. Thus, each node corresponds to a combined assignment of plain text to cipher text at each of the nodes in the path from the given node to the root. A score associated with each node is its depth minus the number of times a right-hand branch was taken in its path from the root. For leaves, the score corresponds to the number of words to be maximized in the criterion (6). Although the number of nodes in the tree can be quite large, and grows exponentially with the number of words in the cryptogram, generally only a very small portion of this tree needs to be explored.

We apply a left-to-right, depth-first search starting from the root which corresponds to a totally blank partial permutation. As we move down the tree, we construct a partial permutation at each node, noting which character mappings are required by the word assignments in its path from the root. At each branch corresponding to a word of a pattern set, blank entries in the permutation are further specified compared to the parent node. Pruning occurs whenever a new entry would be incompatible with the partial permutation. At the rightmost branch in each set, corresponding to a word not in the dictionary, the parent's partial permutation is simply copied to the child.

To check consistency, the algorithm builds up the permutation incrementally as each assignment is made, keeping track of a 26-component vector and its inverse. As cipher words are given plain-word assignments in the various branches, the blanks are filled in. Whenever a word assignment is considered, the algorithm checks in the partial permutation constructed so far that the cipher letters of the cipher word have not already been assigned to other plain letters. It also checks that the plain characters have not already been assigned to other cipher characters. If no inconsistencies are found, the resulting (less partial) permutation is assigned to the child node.

Because of the exponential growth of the tree, it is important to prune as high as possible. So, the order in which the cipher words are assigned to levels is very important. Three somewhat conflicting heuristics that seem natural for this ordering are:

a. Cipher words with short pattern sets should go first, so that the longer sets are left for deeper levels, where they are more likely to remain unvisited due to prunings above them. This heuristic corresponds to what many experienced puzzle solvers do. They begin by seeking out cipher words and repeated letters, such as YJSY and NSMSMS which suggest the plain-text words THAT and BANANA (or ROCOCO).

b. Cipher words which have letters in common with cipher words at higher levels should go above cipher words with no letters in common with higher levels, since they are likely to have most words in their pattern set conflict with the parent permutation and hence be pruned.

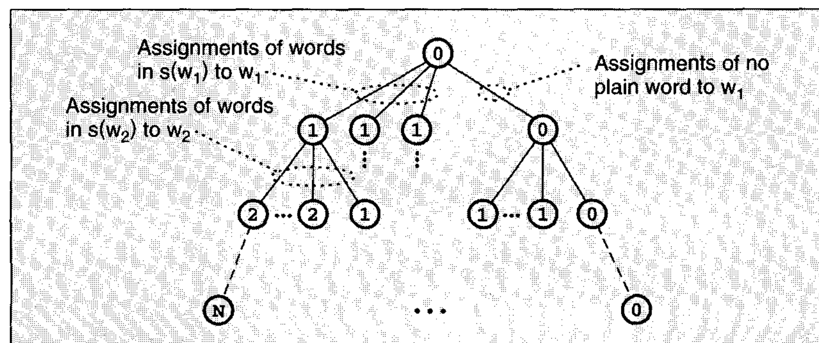
c. Longer words (or words with more distinct characters) should go first, so they force more entries in the partial permutation, thereby setting up a greater likelihood of pruning in the next level down.

A combination of these heuristics has been implemented and found to work satisfactorily. For the top-level word, we choose the cipher word for which the number of distinct letters divided by the size of its pattern set is maximum, i.e., heuristics (a) and (c). For each lower level, we then choose the cipher word having the largest number of distinct letters in common with all the cipher words above it. The order is computed once, before the search begins.

The algorithm consists of the following steps:

1. Input the cipher text and parse it into words.[2]

Figure 2. Pattern sets of words from the 135-word dictionary which may be decipherings of the given cipher words. While dozens of words match the general 2-letter and 3-letter patterns, few possibilities exist for fumost longer words or words with repeated letters. The only 4-letter word from this dictionary in which the 4th letter repeats the 1st is "THAT," and there are no 8-letter words.



2. Group the cipher words by pattern, dropping repetitions.
3. Scan the dictionary and construct the pattern sets of plain words for each cipher word. Ignore any for which the pattern set is empty.
4. Order the cipher words using one or more of the heuristics (a,b,c) given earlier.
5. Call SOLVE (1,*f_null*,0), described later.
6. END.

The first three steps are already completed in Figure 2. The MATCH? function preceding is used for steps 2 and 3. Step 5 invokes a recursive procedure, SOLVE, which explores the tree. Its three arguments are the depth in the search tree, the partial permutation inherited from the parent node, and the number of plain words used so far from the dictionary, i.e., the score of Figure 3.

```
PROCEDURE SOLVE (Depth, f, Score) {
  IF Depth > Dmax THEN COMPARE_SCORE ELSE {
    FOR x ∈ s(w_i) DO
      IF CONSISTENT?(f, w_i, x) THEN SOLVE(Depth + 1, f_new,
        Score + 1);
    SOLVE(Depth + 1, f, Score) };
  RETURN
```

The first line of this procedure just stops the procedure at the leaves of the tree, where a procedure COMPARE_SCORE is called. *Dmax* is the depth of the leaves, determined in Step 3. COMPARE_SCORE should compare the score to the best found so far and output $f^{-1}(Z)$ if the score is at least as good as the best found yet. The remainder of SOLVE simply checks if the permutation, f, can consistently be extended by having the cipher word for this depth be a coding of any plain word, x. For those plain words, the deeper levels are explored by recursive calls to SOLVE. As a side-effect of checking for consistency, the variable *f_new* (local to SOLVE) is set up with the extended permutation. The last line explores the rightmost branch under the assumption that the word for this depth is not in the dictionary. So in that case, the score of the parent is not incremented.

The subroutine to extend and test the partial permutation is as follows. Here, the permutation f is stored as a structure containing two 26-component vectors: an enciphering permutation, *f.c*, and its inverse, the decoding permutation *f.d*. The symbol '*' marks unspecified entries.

```
FUNCTION CONSISTENT? (f, W, X) {
  f_new = f;
  FOR I = 1 TO Length(W) DO {
    IF f_new.c(X[I]) ≠ '*' AND f_new.c(X[I]) ≠ W[I] THEN RETURN
      FALSE;
    IF f_new.d(W[I]) ≠ '*' AND f_new.d(W[I]) ≠ X[I] THEN RETURN
      FALSE;
    f_new.c(X[I]) = W[I];
    f_new.d(W[I]) = X[I] };
  RETURN TRUE }
```

The output of the algorithm is a listing of successively higher scored output texts. The first time a leaf is reached, generally only a few words are matched. Later solutions are only printed if they contain at least as many matched words as the best solution yet printed. The final line(s) of output contain(s) the maximum-likelihood estimate(s) of the plain text.

The size of the dictionary is one of the parameters of the algorithm. There is nothing special about the 50% point used to determine the number 135 for (2). In general, larger dictionaries are preferred. It is, then, more likely that the algorithm will zero in on the exact solution, although it may run slower.

An optional modification is to block those permutations in which a letter is mapped into itself. It is an unwritten rule of recreational cryptograms that each letter is mapped into a different letter, never itself. This is easily implemented as an additional test in the subroutine CONSISTENT?. We prefer not to do this, however, for two reasons. First, it would eliminate a simple way of exercising the algorithm. Because our algorithm is "permutation invariant," we can test it with plain text used as cipher text. We get the same output as if the input were coded, because it is as difficult for the algorithm to find the identity permutation as any other permutation. Second, it allows us to apply the algorithm to ciphers using numbers, pictures, or other symbols as cipher text, by applying first an arbitrary one-to-one mapping from the symbols to the alphabet.

[3]The threshold is more or less than 30 if we take into account the facts that cryptogram sentences are often purposely chosen to be "difficult" and so would have a higher source entropy than random English samples, and also that we are dealing with a partial permutation having less than 26! possibilities. Shannon [8] also argues that cryptograms of about this length are the most difficult to solve, since they maximize his "work characteristic," which is an intuitive notion of computational time complexity.

Figure 3. Word search tree, showing the score at each node. Branches correspond to assigning plain words to cipher words. Most of the tree is not visited, since subtrees corresponding to inconsistent letter assignments are pruned.

Given Cipher Text: YP NR, PT MPY YP NR: YJSY OD YJR WIRDYOPM.

Cipher words:	YP, NR, PT, OD	MPY, YJR	YJSY	WIRDYOPM
Pattern-sets of plain words:	OF, TO, IN, IS, HE, IT, AS, ON, BE, AT, BY, OR, AN, WE, NO, IF, SO, UP, DO, MY, ME, MR	THE, AND, WAS, FOR, HIS, HAD, NOT, ARE, BUT, ONE, YOU, HER, SHE, HIM, HAS, WHO, OUT, ITS, CAN, NEW, TWO, MAY, ANY, NOW, OUR, MAN, WAY, HOW, OWN, MEN, GET, DAY	THAT	(none)

We also employ a bounding technique to prune the search tree under conditions where the algorithm cannot possibly beat the best score yet found, even if all lower levels were to result in a match. The first line of SOLVE becomes "**IF** (*Depth* > *Dmax*) **OR** (*Score* + *Dmax* − *Depth* + 1 < *HighScore*) **THEN** . . ."

Results

The algorithm has been implemented in C on an IBM PC compatible, tried on over a hundred published cryptograms, and generally provides a readable answer in a fraction of a second. Occasionally—on perhaps 5% of the examples tried—it requires a minute or two. This is understandable for a method based on depth-first search. Improved word-ordering heuristics might reduce the fraction of such cases.

Very short cryptograms, of less than 30 characters, are often ambiguous, and so the algorithm gives several tied solutions. Most of these can be eliminated by the user as not making grammatical sense. The following six solutions result for the first example using a 1,000-word dictionary (which then includes the word QUESTION).

TO BE, OF NOT TO BE: THAT IS THE QUESTION.
TO WE, OF NOT TO WE: THAT IS THE QUESTION.
TO ME, OF NOT TO ME: THAT IS THE QUESTION.
TO BE, OR NOT TO BE: THAT IS THE QUESTION.
TO WE, OR NOT TO WE: THAT IS THE QUESTION.
TO ME, OR NOT TO ME: THAT IS THE QUESTION.

The algorithm does not have enough information to determine that only the fourth of these choices is of interest, because it relies on word counts, and uses no grammatical information. If it were required that the ties be broken automatically instead of by eye, grammatical information or other principles for refining the probability measure $P(S)$ could be incorporated into the method. (Using the heuristic ordering principles described earlier, these six solutions are directly constructed, with no backtracking from suboptimal branches, and no exploring of the rightmost branches, which are pruned by the bounding technique.)

It is conceivable for a given cryptogram and dictionary that a pseudosolution could be found in which an incorrect permutation gives a higher word count than the correct permutation. In much experimenting with the algorithm, this has not yet occurred with any sentence longer than two words. This is consistent with Shannon's notion of "unicity distance" [8, 10]. It can be shown that roughly 30 characters of English are enough to expect a unique solution, while shorter cryptograms are inherently ambiguous. The length of the example is close to the threshold for uniqueness.[3]

When the cipher text contains letters not found in any matched word, the partial permutation does not specify the output text completely, and the algorithm prints a nonalphabetic character ('*') to indicate that a letter cannot be determined. For example, with the 135-word dictionary, not containing QUESTION, there is no way for the algorithm to determine the missing letters QU, so the reader must fill it in by eye:

TO BE, OF NOT TO BE: THAT IS THE **ESTION.

The ESTION portion of the word is determined because these letters appear in other words of the sentence. A simple extension with a "spelling checker" algorithm could be employed to fill these gaps if a full dictionary is available. However, experience shows that the user can, trivially, fill these in by inspection, so this is counted as a complete solution.

A typical result for a published cryptogram [12] is:

I SHOOT THE HI**O*OTAMOUS WITH BU**ETS MADE OF **ATINUM BECAUSE IF I USE *EADEN ONES HIS HIDE IS SURE TO F*ATTEN 'EM

One test of special interest is a sentence with a very unusual letter distribution, which confuses cryptographic approaches based on letter n-tuple samples. An example near the limits of the method is [12]:

*OCO HOBO ONCE HAD **AU WHO PUT HOI PO**OI INTOCOMA WITH AN A*IA MADE *AMOU* BY BOY *OP*ANO

The missing letters are L, F, R, and S. Although the performance is marginal—several minutes are required, and many blanks remain—this example would be impossible for a statistical method to solve, which would be thrown off by the large number of word-final vowels used purposely. An even more extreme example comes from Ernest Wright's *GADSBY,* [13] a 267-page novel with no occurrences of the letter E. The output (using the 1,000-word dictionary), is quite readable, despite the highly unusual letter distribution:

UPON THIS BASIS I AM GOING TO SHOW YOU HOW A BUN*H OF BRIGHT YOUNG FOL*S DID FIND A *HAMPION; A MAN WITH BOYS AND GIRLS OF HIS OWN;

The method would fail completely if no words of the plain text were in the dictionary. But, for an n-word sentence this happens with probability 2^{-n} using the 135-word dictionary, and with even lower probability using larger dictionaries.

Conclusion

A simple method has been presented for solving cryptograms and found to work well—even in difficult cases where only a short sample of text is available with letter probability distributions far from what would be expected. Of course, it also works on longer and easier cryptograms. It executes faster and requires much less text than a relaxation method based on a letter-triple distributions—the only other published algorithm for solving cryptograms known to this author. Although exponential time is required in the worst case, in practice it is quite fast. Only a modest amount of storage is required.

[3] Words with internal apostrophes are not handled here, but it is easy to treat them properly: apostrophes are considered letters which the permutation may not change.

PROVEN,
READY-TO-USE
Object-Oriented
TOOLS

See us at OOPSLA in Portland Booth #124!

C++ POWER PARADIGMS
Mark Watson
0-07-911787-2 / 250 pp. / $32.95 (Paperback/disk)
Shorten development time and create viable object-oriented programs with ready-to-use C++ classes and libraries on disk.

C++ PRIMER FOR C PROGRAMMERS
Second Edition
Jay Ranade and Saba Zamir
0-07-051487-9 / 484 pp. / $34.95 (Paperback)
Tap the power of C++ in object-oriented design with this expanded bestselling guide—now based on Borland's C++ 4.0 compiler.

SOLVING THE PRODUCTIVITY PARADOX
TQM for Computer Professionals
Jessica Keyes
0-07-034476-0 / 300 pp. / $40.00 (Hardcover)
Make your computer investments pay off by using the latest total quality management techniques to implement high technology.

THE OBJECT-ORIENTED ENTERPRISE
Making Corporate Information Systems Work
Robert Mattison and Michael J. Sipolt
0-07-041031-3 / 400 pp. / $45.00 (Hardcover)
Create, maintain, and manage effective object-oriented databases in a corporate environment with this step-by-step design guide.

OBJECT-ORIENTED CLIENT/SERVER APPLICATION DEVELOPMENT
Using ObjectPAL and C++
Steve Ayer
0-07-002861-3 / 276 pp. / $45.00 (Hardcover)
Improve your skills with techniques for the analysis, design, and development of a C/S-based application.

OBJECT-ORIENTED DATABASES
Technology, Applications, and Products
Bindu R. Rao
0-07-051279-5 / 350 pp. / $40.00 (Hardcover)
Profiles three commercially available products—ODI/ObjectStore, Objectivity/DB, and Versant—to help you create your own OODBMS.

Available at your local bookstore,
or call toll-free 1-800-352-3566
On CompuServe: GO MH
On Internet: 70007.1531@compuserve.com

McGraw-Hill

Acknowledgement
I would like to acknowledge the programming assistance of Jian Wang.

References

1. Ball, R. *Mathematical Recreations and Essays.* Macmillan, New York, 1960.

2. Friedman, W.F. *Elements of Cryptanalysis.* Aegean Park Press, Laguna Hills, Calif., 1976.

3. Gaines, H.F. *Cryptanalysis.* Dover, New York, 1956.

4. Hunter, D.G.N. and McKenzie, A.R. Experiments with relaxation algorithms for breaking simple substitution ciphers. *Comput. J. 26* (1983), 68–71.

5. Konheim, A. *Cryptography: A Primer,* Wiley, New York, 1981.

6. Kucera, H. and Francis, W.N. *Computational Analysis of Present-Day American English.* Brown University Press, Providence, R.I., 1967.

7. Peleg, S. and Rosenfeld, A. Breaking substitution ciphers using a relaxation algorithm. *Commun. ACM 22* (1979), 598–605.

8. Shannon, C.E. Communication theory of secrecy systems. *Bell Syst. Tech. J. 28* (1949), 656–715.

9. Sinkov, A. *Elementary Cryptanalysis: A Mathematical Approach.* Singer, 1968.

10. van Tilborg, H.C.A. *An Introduction to Cryptology.* Kluwer, New York, 1988.

11. Van Trees, H.L. *Detection, Estimation, and Modulation Theory.* Wiley, New York, 1968.

12. Williams, E.A. *An Invitation to Cryptograms.* Simon and Schuster, New York, 1959.

13. Wright, E. *GADSBY.* Wetzel Publ. Co., Los Angeles, Calif., 1939.

About the Author:
GEORGE W. HART is currently an associate professor at Columbia University in the department of electrical engineering and a member of the Center for Telecommunications Research, working in the areas of systems and communications. **Author's Present Address:** Department of Electrical Engineering, Columbia University, New York, NY 10027-6699; email: hart@ctr.columbia.edu.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

©ACM 0002-0782/94/0900 $3.50