1987

Decoding Substitution Ciphers by Means of Word Matching with Application to OCR

George Nagy
Rensselaer Polytechnic Institute, nagy@ecse.rpi.edu

Sharad C. Seth
University of Nebraska-Lincoln, seth@cse.unl.edu

Kent Einspahr
University of Nebraska-Lincoln

Decoding Substitution Ciphers by Means of Word Matching with Application to OCR

GEORGE NAGY, SHARAD SETH, AND KENT EINSPAHR

Abstract—**A substitution cipher consists of a block of natural language text where each letter of the alphabet has been replaced by a distinct symbol. As a problem in cryptography, the substitution cipher is of limited interest, but it has an important application in optical character recognition. Recent advances render it quite feasible to scan documents with a fairly complex layout and to classify (cluster) the printed characters into distinct groups according to their shape. However, given the immense variety of type styles and forms in current use, it is not possible to assign alphabetical identities to characters of arbitrary size and typeface. This gap can be bridged by solving the equivalent of a substitution cipher problem, thereby opening up the possibility of automatic translation of a scanned document into a standard character code, such as ASCII. Earlier methods relying on letter *n*-gram frequencies require a substantial amount of ciphertext for accurate *n*-gram estimates. A dictionary-based approach solves the problem using relatively small ciphertext samples and a dictionary of fewer than 500 words. Our heuristic backtrack algorithm typically visits only a few hundred among the 26! possible nodes on sample texts ranging from 100 to 600 words.**

Index Terms—**Cryptograms, dictionary based solution, heuristic search, optical character recognition.**

I. Introduction

A substitution cipher consists of a block of natural language text (English) where each letter of the alphabet has been replaced by a specific symbol. The symbol might be another letter, a number, or an arbitrary ideograph, but the mapping from the letters to the symbols must be one-to-one.

Solving or decrypting a cipher means determining the mapping. We develop a family of algorithms to perform this task; by extending a tentative assignment of letters to symbols according to the degree of match between the decrypted portion of the ciphertext and a vocabulary of common words. Our goal is to decrypt a relatively short segment of text, less than one typed page, using a small dictionary of a few hundred words, with computing resources equivalent to a few seconds of CPU-time on a microcomputer.

Our motivation for solving substitution ciphers came from optical character recognition. As optical scanners have become less and less expensive, it is now possible to envision personal computer attachments that can read small batches of text, each in a different format and in a different typeface [1]–[4]. Recent advances render it quite feasible to scan documents with a fairly complex layout and to classify (cluster) the printed characters into different groups according to their shape [5], [6]. Given, however, the immense number of typestyles in current use, it is not possible to assign alphabetical identities to characters of arbitrary size and typeface. Whereas the number of different shapes encountered in a typical business or engineering document is only of the order of a few hundred, there are tens of thousands of symbol shapes commonly used by the printing industry. With the increasing availability of economical high-resolution dot-matrix printers and photocomposers, this diversity is more likely to increase than decrease: given convenient facilities, many of us will want to design—like Knuth—our own typefaces. An alternative to conventional OCR methods that determine alphabetic identities is to solve a substitution cipher where the plaintext consists of the scanned copy, and the ciphertext consists of the arbitrary codes assigned to each letter by the clustering program.

The feasibility of building such a character recognition system has been demonstrated more than a decade ago [7]–[9]. At that time, however, on-line dictionary look-up seemed impractical, so we used methods based on letter frequencies. This required segments of texts consisting of several thousand words and a large, dedicated computer. Since then, the application of relaxation methods has resulted in improved *n*-gram frequency algorithms [10], [11]. In this correspondence we capitalize on advances in search algorithms, string matching, and dictionary structures to solve substitution ciphers using dictionary look-up.

II. Statement of the Problem

In the idealized formulation of the problem the ciphertext and plaintext alphabets are of the same size and are denoted, respec-

Manuscript received September 30, 1985; revised January 1986. Recommended for acceptance by T. Pavlidis. This work was supported by the National Science Foundation under Grant DCR-8421162.

G. Nagy is with the Department of Electrical, Computer, and Systems Engineering, Rensselaer Polytechnic Institute, Troy, NY 12180.

S. Seth and K. Einspahr are with the Department of Computer Science, University of Nebraska, Lincoln, NE 68588.

IEEE Log Number 8715508.

tively, by C and P. The (unique) solution is a one-to-one mapping from C to P.

The elements of the ciphertext alphabet will be called *symbols*, and those of the plaintext alphabet, *letters*. In both the plaintext and the ciphertext there will be a definite reading order of words but this is of no consequence in our formulation. Neither is the fact that a particular word occurs more than once in the text. That is, we consider the ciphertext as a finite set of distinct words (each of which itself is a finite string of symbols). We also assume the availability of a plaintext *dictionary* which is another set of words comprised of letters. The idealized problem, thus formulated, is completely symmetric with regard to the roles of the dictionary and of the ciphertext: the objective is simply to find a mapping between the alphabets of two sets of unique words in such a way as to maximize the correspondence between the two sets.

The solution to the problem can be built up incrementally starting with the null solution. At each step, one or more new elements of the mapping from C to P are added from amongst the remaining choices. The selection of mapped pair(s) is guided by both the given ciphertext and dictionary, the aim being to minimize the expected search time. Without guiding heuristics for selecting symbol-letter pairs, the correct assignment requires an exhaustive search of the solution space which will be impractical for all but the most trivial problems.

Since search length will be used in evaluating competing heuristics, it is important to introduce a measure of search length which is unbiased and general-purpose in its applicability, e.g., the count of comparisons made at the character level.

The ideal formulation of the problem considered in this paper would have to be relaxed in several ways to cope with the complexities of the problem encountered with "real-world" data. For example, in practice, the symbol classification mechanism is not likely to be perfect; two occurrences of the same symbol may be classified as being different because of imperfections on the printed surface, variations in ink-deposition, smears, etc. This would invalidate the assumption of one-to-one mapping from symbols to letters; instead, we must consider the more general situation of a many-to-one mapping. Many-to-one mapping would also result from the use of upper/lower case in the ciphertext while restricting the dictionary to be single-case for computational efficiency, or from the use of boldface and italics. A less likely situation is that of two distinct symbols identically classified, again, due to a variety of reasons (e.g., inadequacy of classification algorithm, undifferentiated letter-pairs such as I-l in some typefaces, erasure of crucial distinctive features in the printing or scanning process). Thus, in the most general case, we may not even have a functional relationship from symbols to letters.

Lastly, the mapping from symbols to letters may not be complete; there may be symbols in the ciphertext, such as punctuation, which do not occur in a spelling dictionary. This could also happen with dictionaries that are too short to include all the letters. Conversely, there may be some less frequently occurring letters, such as "j", which do not occur at all in a short ciphertext.

III. Search Algorithms

We will use the following general form of a solution to the substitution cipher problem as the point of departure for further discussion.

```
algorithm TREE-SEARCH(input, output)
  {TREE-SEARCH solves the substitution cipher by selecting a
  group of ciphertext words to match to the dictionary and extending
  the assignment set on the basis of matched letter frequencies.
  A cutoff strategy is used to detect a "bad" assigment early and
  backtrack to the level at which such assignment was made.}
  INITIALIZE;
  EXTEND(PA, level, CIPH.count, DICT.count);
    {The current partial assignment PA is extended. This is a recursive procedure resulting in a depth-first search of the possible solutions to the substitution cipher. The last three parameters are used to implement cutoff. A detailed description of EXTEND follows.}
    if COMPLETE(PA) then [record PA; return]
      {PA is assumed to be complete if all the ciphertext symbols
      which also occur in the matched words in the dictionary are
      assigned. Since a unique solution is expected, the return
      may be replaced by a super return from all the levels of
      recursion.}
    else
      RANKSYMBOLS(PA, x);
        {Produces an ordered list x of vectors x' of unassigned
        symbols.}
      for each x' in x do
      RANKLETTERS(PA, x', y);
        {Produces an ordered list y of vectors y' of letters each
        of the same length as x'}
      for each y' in y do
        ASSIGN(x', y', PA, PA');
          {y' is assigned to x' thereby extending PA to PA'.
          Also updates CIPH.count and DICT.count.}
          if level < T3 then CUTOFF(T1, T2);
            {CUTOFF checks if x' occurs at least T1 times in
            its selected subgroup and DICT.count/CIPH.count
            is less than T2. If both are true, backtrack to an earlier level occurs.}
          if not cutoff then do
            level ← level + 1;
            EXTEND(PA', level, CIPH.count, DICT.count)
            level ← level-1;
          end (if);
        end (for each y')
      end (for each x');
  end (EXTEND);
end (TREE-SEARCH).
```

EXTEND can be thought of as implementing a depth-first tree search. Between successive nested calls to EXTEND, choices are made, first of vector x' of symbols and then a vector y' of letters, the two choices together defining a single level of the standard search tree. RANKSYMBOLS and RANKLETTERS allow heuristic reordering of the search [12]–[14].

A. Bestfirst Heuristic

The first heuristic to be discussed attempts to extend the current partial assignment PA by considering only a subset S of ciphertext words which have just one symbol unassigned (if no such words exist then we consider those with two symbols unassigned, etc.). The end result is a list of strings (vectors) x' of symbols each of the same length (one or more) and each derived from a ciphertext word. RANKSYMBOLS produces this list and reorders it in decreasing order of the frequency of each x' in S. The reordered list is called x in the algorithm and its elements are selected in order by the algorithm. For a particular choice x', each ciphertext word in which only the elements of x' appear unassigned is considered for all possible matches in the dictionary. Whenever a match occurs there is a potential extension of the current partial assignment since the matched dictionary provides a unique mapping for symbols in x'. The extension can be conveniently denoted by a string y' of letters of the same length as x' where each symbol in x' is mapped to the corresponding element of y'. RANKLETTERS finds all possible strings y' associated with a particular choice of x' and reorders them as follows. Consider the mapping from x' to a particular choice of y'. For each such choice, one or more of the ciphertext words which had only the elements of x' as unassigned, become completely assigned. RANKLETTERS records a count of the number of these found in the dictionary as a measure of goodness of the particular choice of y' and reorders these choices according to the decreasing value of this count. The reordered list is called y in the algorithm.

Backtracking to the previous level of the tree occurs when no assignment can be made at the current level (the x and y lists are exhausted at the current node). However, our experience with some simple examples points up the need for a more powerful backtracking mechanism. Thus a second heuristic mechanism, called *cutoff*, is invoked to allow backtracking many levels up the search tree.

Because even correct words may be missing from our small dictionary, the cutoff procedure must be statistical in nature. Furthermore, we not only wish to determine whether the entire assignment vector is correct but, if it is not, which assignment is most likely to be invalid. Consequently, we keep track both of the number of completed ciphertext words in which each symbol appears and the number of these words that appear in the dictionary. If the number of text words in which a given symbol appears in sufficiently large, and the number of these words which are found in the dictionary is sufficiently low, then we backtrack to the point where the suspect assignment was made.

The cutoff procedure does not require any additional dictionary lookups because any completed ciphertext word must already have been checked against the dictionary when its last symbol was assigned.

While the cutoff procedure works well on correcting misassignments of high-probability letters, it results in unjustified backtracking later in the process when low-probability letters are assigned. This is not surprising, since the low-probability letters tend to appear in fewer common words. Furthermore, late in the assignment process cutoff saves less time, since even complete exploration of the bottom part of the tree is computationally quite feasible. We therefore added a third parameter, which disables the cutoff procedure once a certain number of assignments have been made. This may be regarded as an approximation to an adaptive cutoff mechanism that takes into account the progress of the search.

Example: For this example we chose a ciphertext from the *Datamation* magazine consisting of a total of 651 words and a dictionary of the 442 most frequently used words of at most six letters. The ciphertext is reduced to a word list in lower-case only before its use by the algorithm. Ignoring capitalization allows one to consider the simplified case of a one-to-one mapping between symbols (ciphertext symbols) and letters (plaintext symbols). For all examples we will follow the convention that lower-case represents unassigned symbols and upper-case represents the letters.

Fig. 1 shows an example of the Tree-search algorithm for one level of recursion. At the time when we enter the level-3 node in Fig. 1 it is assumed that the Tree-search algorithm has already made the (correct) assignments shown to the symbols "t", "a", and "l". We will walk through the steps of the algorithm until the next assignment is made.

The algorithm first selects the group of ciphertext words which have the fewest number of unassigned symbols. In this case (as in all cases we have examined to date) it was possible to choose words with just one unknown. The unknown symbols are ranked according to the counts of their occurrence in this group of words. Thus the unknown "o" which occurs most frequently in the group (four times) is selected first for assigning a letter.

Next, the subgroup of words in which the selected "o" appears is matched against the dictionary. In this case "to" and "too" will match correctly with "TO" and "TOO" but "lot" will match incorrectly with "LET". However, the incorrect assignment "o" → "E" is deferred in favor of the correct assignment "o" → "O" based on the counts of "O" and "E" (respectively, 3 and 1) in the matched dictionary words.

The cutoff heuristic in this example is based on the accumulated counts (DICT.count and CIPH.count) shown at the bottom of Fig. 1. The values existing at the time of entry to the level-3 node are updated after the recursion step traced above. The increments reflect the number of occurrences of the symbols and the letters in the selected groups of ciphertext and dictionary words. These accumulated totals are used to determine whether the last assignment has a high probability of being "bad." A "bad" assignment occurs when DICT.count/CIPH.count is below a preset threshold. If so, the algorithm cuts off to the level of recursion at which the 'bad' assignment was made. Actually, three thresholds are used to control the cutoff procedure. The first threshold, $T1$, determines how many occurrences of a specific symbol need to occur before cutoff is checked. The second threshold, $T2$, discussed above, causes a cutoff only if the symbol occurrences exceed the first threshold. The final threshold, $T3$, is the recursion level at which the check for cutoff is terminated. For the example shown in Fig. 1 the three threshold values used were 3, 0.5, and 12. With these values, cutoff will be checked for the assignment just traced since 1) "o" occurs more than 3 times in the selected group of ciphertext words and 2) the recursion level is below 12. However, cutoff will not occur because the proportion DICT.count/CIPH.count for each symbol is at least 0.5.

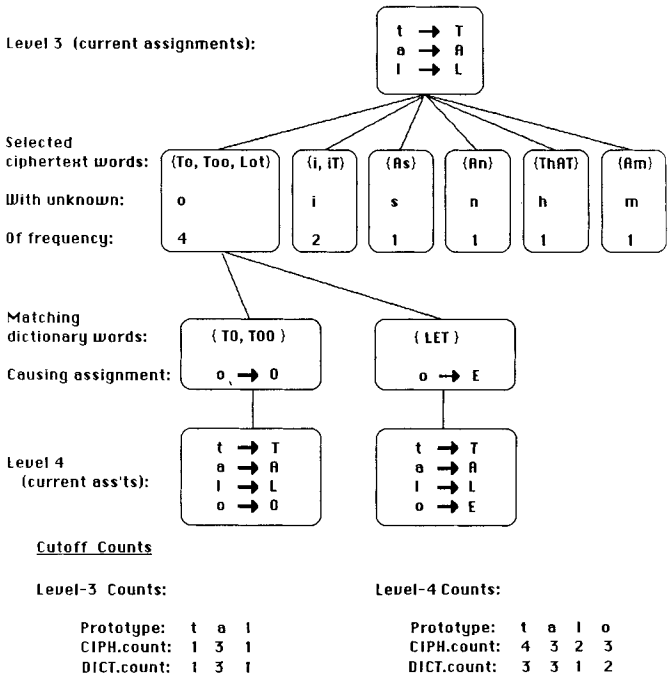

Fig. 1. An example to illustrate the Bestfirst heuristic.

B. Coverset Heuristic

Unlike the Bestfirst heuristic, the Coverset heuristic tries to match longer words first. Only ciphertext words which are not completely defined by the current assignments are retained for further consideration. Let this set be denoted by S. A word w in S is said to *cover* a word v if every unassigned symbol in v is also in w. RANKSYMBOLS orders the ciphertext words in S according to the number of words covered by a word. Now suppose a word w is chosen from S and matched against a dictionary word of the same length. If the match is successful, it will result in assignment of letters to all the unassigned symbols in both w and the words covered by w. The latter can be looked up in the dictionary and a count kept of the number found. RANKLETTERS orders the potential matches for w according to the value of this count.

Example: Fig. 2 shows one level of recursion for the Tree-search algorithm using the coverset heuristic. Since the ciphertext word "there" covers the greatest number (4) of words, it will be ranked first by RANKSYMBOLS. The dictionary words that are possible matches to "there" are "THERE" and "THESE". RANKLETTERS selects the matched word that maximizes the number of coverset words which also found in the dictionary. If "there" is assigned to "THERE", four words can be matched whereas only three coverset words are matched if "there" is assigned to "THESE". Each unassigned symbol of the ciphertext word is assigned to the corresponding letter in the matched dictionary word ("h" to "H" and "r" to "R"), extending the current assignment. The accumulated counts (DICT.count and CIPH.count) shown at the bottom of Fig. 2 are derived in the same way as those in Fig. 1.

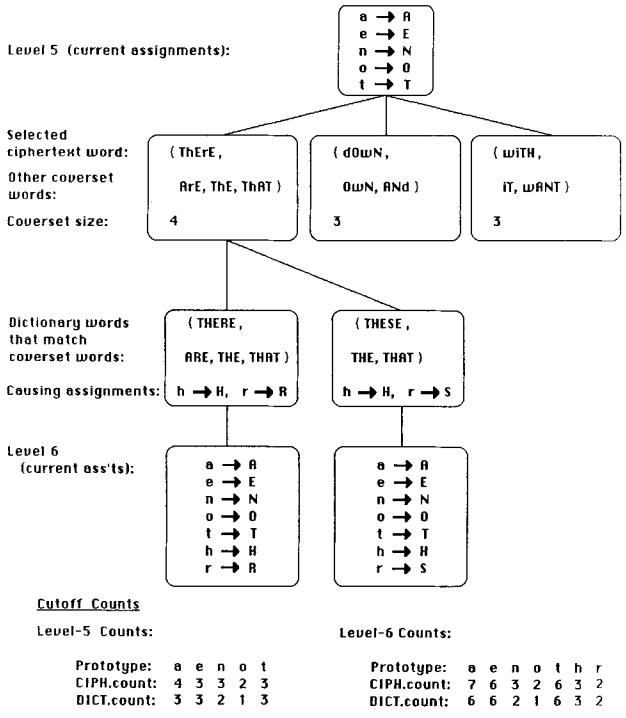

Fig. 2. An example to illustrate the Coverset heuristic.

IV. Experimental Results

The algorithms described above were programmed in Pascal and a series of experiments were run on VAX 11/780 and PDP 11/70 computers with the following objectives:

1) Determine the size of the dictionary necessary to decipher relatively short segments of texts.

2) Investigate the improvement resulting from the cutoff heuristic over the original greedy search algorithm, and compare several versions of the cutoff heuristic.

3) Check the robustness of the algorithm by applying it to text segments of diverse length and origin, and explore the range of parameters that yield useful results.

4) Demonstrate the algorithm on a problem solved by the relaxation technique [10] and compare performance.

The characteristics of the sample texts used in the experiments are shown in Table I. Since our dictionaries do not contain any words longer than 6 letters, all longer ciphertext words are discarded. Accordingly, Table I shows for each sample text the total number of ciphertext words, the number of *distinct* ciphertext words of six letters or less, and the number of these that are found in the dictionary. We also show what symbols are missing entirely from the text, and what symbols occur *only* in words that are not in the dictionary. The last column shows the number of symbols that can be correctly assigned.

In reporting results, we must consider the various possible outcomes of applying the algorithm to a sample of text. There are four possibilities:

1) The algorithm does not assign a letter identity to every possible symbol within the amount of CPU time available (about 200 seconds on the PDP 11/70, which corresponds to about 250 000 attempts to match a ciphertext word in the dictionary).

2) The algorithm is terminated before assigning letters to every symbol. This happens when there are no ciphertext words left (with one or more unassigned symbols) for which a match can be found in the dictionary, and threshold $T3$ has been exceeded, disabling backtracking.

3) The algorithm completes assigning letters to every symbol. Some of the assignments may, of course, be incorrect; most often these are infrequently occurring symbols such as j or q.

TABLE I
Characteristics of Sample Text Segments

Code†	Word Count	Distinct Usable Words	Dictionary Matches Count	Prototypes not in Dictionary	Prototypes not in Ciphertext	Number of Prototypes Assignable
SP	122	64	45	-	j p x j	22
LP	329	134	97	x	z	24
DA	651	170	110	j q	z	23
AD	95	43	34	-	j q x z	22
ED	397	163	105	q	z	24
NW	333	132	79	q	x z	23
GA	272	107	72	q	j x z	22

†SP: Short Physics, LP: Long Physics, DA: Datamation, AD: Auto Ad., ED: Royko Editorial, NW: Newsweek, GA: Gettysburg

4) The algorithm completes the search—usually abbreviated by cutoff—without reaching a solution.

A. Dictionary

Dictionaries of sizes 100, 200, and 500 of the most frequently used words were tried on the short physics text. In all experiments the 100-word dictionary produced considerably poorer results in terms of the total number of assignments made. The 200 and 500 word dictionary produced comparable results. The execution times were slightly worse for the 500-word dictionary but this performance degradation was due to the linear organization of dictionaries implemented for its simplicity. We chose the larger size with one modification: all words longer than 6 letters were deleted from this dictionary for reasons of efficiency. The resulting 442-word dictionary was used for all the experiments reported below. To ensure the inclusion of the single word in a short segment of text that might contain a rare letter such as "z", an enormous increase in dictionary size would be necessary.

B. Cutoff

The performance of the algorithm with and without cutoff is shown in Table II. It is seen that cutoff is effective in obtaining a solution in all cases where the search without cutoff times out. However, when the search without cutoff converges rapidly, cutoff may entail a time penalty.

C. Robustness

The three threshold parameters were varied for the *Newsweek* text in the range

$$3 < T1 < 5; \quad 0.20 < T2 < 0.50; \quad 6 < T3 < 18.$$

Within this range, the final assignments were identical, with the number of calls to EXTEND varying only from 24 to 38. Generally, lowering thresholds $T1$ and $T2$ further, or increasing $T3$, results in inappropriate backtracking and often leads to an erroneous assignment. On the other hand, changing these thresholds in the opposite direction inactivates cutoff and increases the number of iterations.

The performance of the Bestfirst heuristic with two sets of parameters is shown in Table III for all the samples. It is seen that the number of iterations (calls to EXTEND) is not necessarily smaller for short segments. On the other hand, the number of character matches for dictionary access (words sought, with or without success, in the dictionary) tends to grow with the length of the segment. The computing time required, in turn, is roughly proportional to the number of accesses; it is clear that even if we cannot lower the number of iterations, we can decrease the computer time by improving the dictionary access method.

D. Comparison to Relaxation Algorithm

In order to provide a statistically valid comparison, we did not experiment at all with the Gettysburg Address until the final test run. After determining apparently the best cutoff parameter values for the Bestfirst heuristic we finally attempted this run, which was also used by Peleg and Rosenfeld [10]. The algorithm assigned correctly 22 out of the 23 symbols that it could assign properly, given our dictionary. After this run, we found that with most other parameter settings the algorithm performed just as well.

TABLE II
PERFORMANCE WITH AND WITHOUT CUTOFF
(Bestfirst Heuristic)

Text	Number of Calls to Extend		
	Cutoff (T1=4, T2=.5, T3=12)		No Cutoff
	To Worst Proportion	To Highest Level	
SP	106	69	TO† (192)
LP	152	No Solution	25
DA	68	80	24
AD	67	67	TO (605)
ED	175	94	TO (66)
NW	24	24	24
GA	30	35	TO (113)

† Timed Out

TABLE III
EFFECT OF CUTOFF PARAMETERS ON PERFORMANCE
(Bestfirst Heuristic)

Text	T1=3, T2=.5, T3=18			T1=4, T2=.5, T3=12		
	Calls to Extend	Dictionary Accesses	Results†	Calls to Extend	Dictioanry Accesses	Results†
SP	251	420,388	16/s/t,l,u,g,x	69	30,036	22/-/-
LP	125	92,570	24/-/x	152	108,402	24/-/x
DA	90	102,351	23/-/-	68	62,020	23/-/q,j
AD	72	31,323	22/-/-	67	28,764	19/b/m,k
ED	79	118,676	24/-/q	94	66,378	23/-/q,x
NW	33	48,411	23/-/q	24	15,483	23/-/q
GA	27	24,411	22/-/q	35	23,789	22/-/q

† Number assigned correctly/misassigned/unassigned

E. Dictionary/Ciphertext Inversion

As expected, the algorithm can also reach the correct solution when the text file and the dictionary file are switched. In a test in which the *Datamation* text and the 200-word dictionary were interchanged with no check for cutoff, all 23 symbols that could be assigned were assigned correctly with 24 calls to EXTEND.

F. Observations

The major problem with the cutoff heuristic is that with relatively short texts there are not sufficient words to determine, for each symbol, whether the assignment is correct. Furthermore, when we have a choice of making an assignment among several symbols, we first select the symbol without regard for the number of words that would be matched in the dictionary by the various symbol candidates. This frequently results in selecting a symbol for which a number of ambiguous assignments can be made (multiple dictionary hits), even when there is another candidate available for which an unambiguous assignment based on several words is available.

V. CONCLUSION

We have demonstrated a heuristic algorithm for assigning alphabetic identities to symbols in a textual context on the basis of a small vocabulary of frequent English words. The storage and computing requirements are relatively modest, and the processing could be performed on a microcomputer-based postprocessor for optical character recognition.

The algorithm assigns correct identities to all but a few infrequently occurring symbols on samples of text ranging from 100 to 600 words. Identities are assigned in sequence, and the algorithm backtracks whenever either a satisfactory next assignment cannot be made, or a large proportion of words with already assigned symbols cannot be found in the dictionary. Typically no more than a few hundred nodes are visited in a search tree which could, in the worst case, comprise 26! nodes.

Since the most time-consuming part of the proposed method is seeking partial matches for groups of words in the dictionary, the dictionary search algorithm and the data structure used for the dictionary have a profound effect on overall computational resource requirements.

The most common method of searching for a match in an ordered list, binary search, is not directly applicable when the search words contain wild cards (i.e., unidentified symbols). An obvious shortcut is to sort the dictionary according to word-length. Although most common words are 2–5 characters long, this may, in fact, be the best strategy for short dictionaries of 100 words or less, Other possibilities are having several copies of the dictionary, each ordered without regard for the characters in specific positions (which correspond to the symbols in the search word), and anagram-based organization where the dictionary is sorted according to the constituent letters of each word without regard to their internal order. In view of the duality between the roles of the ciphertext and the dictionary, the same considerations will apply to the ciphertext.

There is also a voluminous literature on string matching and partial string matching [15], [16]. At first blush, we do not expect much of this work to be applicable because the presence of clear word demarcations in the dictionary and the ciphertext renders attempted matches across word boundaries unnecessary. Other dictionary organization techniques, such as root, prefix, and suffix oriented methods [17], [18], character registers [19], abbreviations [20], common substrings [21], weighted Levenshtein metric [22], and hierarchies [23], may be more relevant. Currently, we are focusing our attention on hash coding, where the small overhead in storage is insignificant with our short dictionary.

We are currently trying our methods on data more representative of the output of a real OCR system, including multiple symbols for each letter. A non-one-to-one mapping from symbols to letters presents a serious difficulty for n-gram techniques, but is not expected to be a problem with dictionary look-up. We are attempting a mixed strategy, combining the strengths of both methods. An alternative is to use a large dictionary, such as those designed for spelling correction, to assign the last few "rare" prototypes. Since only a few accesses would be required at this point, the size of the dictionary would not materially increase the running time. Excellent results using a large dictionary were presented in [24].

The major theoretical task facing us is to investigate the relationship between length of dictionary, length of ciphertext, and probability of correct assignment. Clearly, the longer the dictionary, the shorter is the length of ciphertext necessary to guarantee a sufficient number of matching words to correctly identify all of the symbols. Humans, who have virtually instantaneous access to a very large vocabulary, can readily solve very short substitution ciphers. The probabilistic formulation of the problem is difficult because in English (and in other natural languages) the joint probabilities of the letters constituting a word do not correspond to the probability of that word occurring in a segment of text, and both probability distributions affect the performance of the proposed method.

ACKNOWLEDGMENT

The authors wish to thank D. Kreher for an earlier implementation of the algorithm and C. Grimes for experimentation with dictionaries. The help by T. Meyer and S. Lloyd with the preparation of this manuscript is also gratefully acknowledged.

REFERENCES

[1] G. Nagy, "Optical character recognition—Theory and practice," in *Handbook of Statistics II*, P. R. Krisnaiah, Ed. Amsterdam, The Netherlands: North-Holland, 1982, pp. 621–649.
[2] ——, "Optical scanning digitizers," *IEEE Spectrum*, pp. 13–24, May 1983.
[3] G. Nagy and S. Seth, "Hierarchical image representation with application to optically scanned documents," in *Proc. ICPR-7*, Montreal, 1984, pp. 347–349.
[4] T. Stanton, D. Burns, and S. Venit, "Page-to-disk technology: 9 state of the art scanners," *PC Mag.*, vol. 5, pp. 128–177, Sept. 30, 1986.
[5] K. Y. Wong, R. G. Casey, and F. M. Wahl, "Document analysis system," *IBM J. Res. Develop.*, vol. 26, pp. 647–656, Nov. 1982.
[6] R. Casey, S. K. Chai, and K. Y. Wong, "Unsupervised construction of decision networks for pattern classification," in *Proc. ICPR-7*, Montreal, July 1984.
[7] R. Casey and G. Nagy, "Autonomous reading machine," *IEEE Trans. Comput.*, vol. C-7, May 1968.

[8] ——, "Advances in pattern recognition," *Sci. Amer.*, vol. 224, pp. 56–71, Apr. 1971.
[9] L. Bahl, "An algorithm for solving simple substitution cryptograms," in *Proc. IEEE Int. Symp. Information Theory (abstract)*, Ithaca, NY, 1977.
[10] S. Peleg and A. Rosenfeld, "Breaking substitution ciphers using a relaxation algorithm," *Commun. ACM*, vol. 22, pp. 598–605, Nov. 1979.
[11] D. G. N. Hunter and A. R. McKenzie, "Experiments with relaxation algorithms breaking simple substitution ciphers," *Comput. J.*, vol. 26, no. 1, pp. 68–71, 1983.
[12] N. J. Nilsson, *Principles of Artificial Intelligence.* Palo Alto, CA: Tioga, 1980.
[13] E. Rich, *Artificial Intelligence.* New York: McGraw-Hill, 1983.
[14] P. H. Winston, *Artificial Intelligence.* Reading, MA: Addison-Wesley, 1977.
[15] R. Rivest, "Partial match algorithms," *Siam J. Comput.*, vol. 5, Mar. 1976.
[16] C. S. Roberts, "Partial match retrieval via the method of superimposed codes," *Proc. IEEE*, vol. 67, Dec. 1979.
[17] K. S. O'Mara, W. M. Jaworski, and S. Klasa, "On the development of a recursive model of word structure in the English language," in *Applied Systems and Cybernetics V*, G. E. Lasker, Ed. New York: Pergamon, 1980.
[18] M. T. Chen and J. Seiferas, "Efficient and elegant subword tree construction," *Computer Research Review*, Univ. Rochester, 1984.
[19] J. L. Peterson, "Computer programs for detecting and correcting spelling errors," *Commun. ACM*, vol. 23, no. 12, pp. 676–687, Dec. 1980.
[20] C. R. Blair, "A program for correcting spelling errors," *Inform. Contr.*, vol. 3, pp. 60–67, Mar. 1960.
[21] C. N. Alberga, "String similarity and misspellings," *Commun. ACM*, vol. 10, pp. 302–313, May 1967.
[22] T. Okuda, E. Tanaka, and T. Kasai, "A method for the correction of garbled words based on the Levenshtein metric," *IEEE Trans. Comput.*, vol. C-25, pp. 172–177, Feb. 1976.
[23] E. Tanaka, T. Kohashiguchi, and K. Shimamura, "High speed string correction for OCR," in *Proc. ICPR-8*, Paris, 1986, pp. 340–343.
[24] R. G. Casey, "Text OCR by solving a cryptogram," in *Proc. ICPR-8*, Paris, 1986, pp. 349–351.