Contextual Word Recognition Using Binary Digrams

EDWARD M. RISEMAN, MEMBER, IEEE, AND ROGER W. EHRICH, MEMBER, IEEE

Abstract—This paper describes a special-purpose character recognition system which uses contextual information for the recognition of words from any given dictionary of words. Previous techniques that utilized context involved letter transition probabilities of digrams and trigrams. This research introduces the concept of binary digrams which overcomes some of the problems of past approaches. They can be used to extract effectively the "syntax" of the dictionary while requiring very modest amounts of storage. A computationally feasible procedure is described which allows the accuracy requirements of the character recognizer to be relaxed if it is followed by a contextual postprocessor. The modified recognition system is allowed to output several alternatives for each character, while the postprocessor selects the proper string of characters by having access to both the dictionary and the dictionary syntax. A theoretical estimate of the recognition rate is derived, and experimental results demonstrate the ability of the system to achieve low error and rejection rates.

Index Terms—Admissible sequences, binary digrams, context, dictionary, pattern recognition, postprocessor, special-purpose character recognizer, syntax.

INTRODUCTION

THE purpose of a character recognition system is the proper classification of each input character scanned.

There have been a number of attempts to reduce the error rate in the classification process through contextual analysis [1]–[13]. The decision on the character identity is based upon the identity of neighboring characters and, in many cases, letter sequences have been considered to be Markov chains. These approaches often involve the utilization of digram and trigram statistics (i.e., probabilities of letter pairs and triplets, respectively), and even the use of quadgrams has been considered [7], [10].

The use of digrams in contextual pattern recognition has been valuable. The probability of digram occurrences can be used to resolve confusion in the classification of a character in a given position on the basis of the character in the preceding position. However, digrams may not be satisfactory and trigrams may be required to achieve the desired recognition rate. This can create data storage problems; whereas an entire set of digram probabilities over the English alphabet requires only \(26^2 = 676\) values to be stored, trigrams necessitate \(26^3 = 17,576\) data cells, and quadgrams over 456,000. The cost of utilizing data bases of this size becomes significant.

Another disadvantage of previous solutions has been discussed by Duda and Hart [12]. Classifiers have been used to output a set of alternative classifications, with associated confidences, for each character that is input. Some portion of the time, the actual character is not the alternative with the highest confidence. The proper character may still be selected by choosing the string of characters that has the highest overall confidence in context, weighted by the \(a\ priori\) probability of the string. Duda and Hart note that a combinatorial explosion takes place. A string of length ten with four alternatives for each character gives rise to over 1,000,000 possible strings. The combinatorial complexity of this method is the limiting factor in such solutions.

In their work on the use of context in handwritten character recognition, Harmon and Sitar [4], [5] noted that many digram and trigram probabilities associated with English text are zero. They used these probabilities to accomplish error detection, and with a lower accuracy, error correction in ordinary text.

One other procedure deserves mention here. Bledsoe and Browning [2] suggested computing a confidence index for every possible character in every position. Using these indices one can compute a confidence index for every word in a given dictionary, and then the word with the largest may be selected.

As the use of character recognizers increases, it may be advantageous to consider "special-purpose" recognition systems. The general character recognition machine for English text is designed to operate on all possible strings that can be formed from the 26 letters of the English alphabet. 1 This generality is obtained at the expense of a significant additional effort and cost; the error rate must be acceptable without the use of contextual analysis, or at most, with the use of statistics of sets of letters across a large training sample of text. However, there may be many applications where the system is confined to recognition of specified lists of words associated with such items as names of people, cities, and countries, entries on business or inventory forms, etc. In these cases a small percentage of the possible strings must be recognized. Thus, the classification system can utilize the contextual information in each specific problem so that one may design a less sophisticated recognizer that performs comparably to the general pattern classifier.

The character recognition system presented here is to be applied to the classification of lists, or "dictionaries," which contain no more than a few thousand words. The simpler character recognition system we propose will not be re-

---

1 Punctuation marks and other characters are ignored in this simplified discussion.
quired to output the identity of the input character with a specified confidence, nor will it be required to output a set of alternative characters with confidence measures. Rather, the character recognizer required must output only a set of \( S \) alternatives, without confidences, which contains the correct character a high percentage of the time. Unless the character is seriously distorted, as \( S \) grows larger the probability of finding the correct letter in the set increases greatly; in this respect larger values of \( S \) are desirable. The problem that is explored is the ability of a contextual postprocessor to correctly identify the input string from among the many possibilities with an acceptable error and rejection rate, while keeping the computational complexity under control.

Binary digrams are defined to extract the "syntax" of the dictionary in a simple form. All sequences of alternative characters that satisfy the syntax are collected. Each of the sequences is looked up in the dictionary and discarded if it is not present. If only one of the sequences is present, the postprocessor outputs that sequence as the input word. If there is more than one, various decision strategies can be employed which involve a tradeoff between error and rejection rates. A measure for the recognition rate as a function of the dictionary syntax, the number of alternatives output for each character, and the length of the words is developed. Experimental results are presented for simulation of the system with as many as 350 seven-letter words.

**Recognizer Organization**

In this section we present the basic organization of our recognition system which is roughly shown in Fig. 1. The character recognition procedure is similar to existing algorithms except that the character recognizer, whenever it is uncertain about the identity of a character, outputs a set of characters which it determines are candidates for the particular character in question and among which it may be unable to discriminate. Such a set of letters will be referred to as a substitution set. The recognizer output, instead of being a sequence of possibly incorrect characters, is a sequence of substitution sets, each of which hopefully contains the correct character. Thus, the character recognizer need not be very good so long as it produces substitution sets of reasonable size, each of which contains the correct character.

It is the task of the postprocessor to store substitution sets until the naive recognizer determines that a word has been scanned, and then to determine which is actually the correct word. The most naive approach would be to look up in the dictionary every possible word \( a_k a_k a_k \cdots a_k \) which can be formed by taking \( a_k \) from the first substitution set, \( a_k \) from the second, and so on through \( a_k \). As previously noted, a combinatorial explosion takes place; if we had five-letter substitution sets and eight-letter words this would require \( 5^8 = 390 \, 625 \) dictionary lookups. Fortunately, if the dictionary is not too large there is a substantial amount of context among letters of words. One might consider using Markov letter transition probabilities to eliminate those words which are so improbable that they need not be looked up in the dictionary. Since such a scheme is very costly to implement, one looks for a simpler method for using context.

Henceforth, it will be assumed that there are recognizable spaces between characters allowing the postprocessor to be supplied with the length \( n \) of the word just scanned. This allows the dictionary to be subdivided, and it is necessary only to discuss the selection of the correct \( n \)-letter word from the \( n \)-letter word dictionary. In effect, dimensionality reduction has been achieved by transforming the classification process into a set of disjoint problems, each with lower dimensionality than the original problem. Thus, we will assume, without any loss of generality, that the words have fixed length, \( n \). Also, except in an illustrative example, we assume that the characters are taken from the 26-letter English alphabet.

**Admissible Sequences and Digrams**

There is a simple way in which most words constructable from the substitution sets can be eliminated without having to look them up in the dictionary. Suppose that there is a nonzero probability that the letter \( a_i \) occurs in the \( i \)th position of some word and that \( a_i \) occurs in the \( j \)th position of the same word. We record the information by placing a 1 in the \( (k, l) \)th position of a \( 26 \times 26 \) binary matrix, \( d_{ij} \), called a binary digram. Thus, \( d_{ij}(k, l) = 1 \) implies that some word in the dictionary has \( a_i \) in position \( i \) and \( a_l \) in position \( j \). Notice that since \( d_{ij}(k, l) = d_{ji}(l, k) \) it is never necessary to compute both \( d_{ij} \) and \( d_{ji} \), and we will assume that if \( d_{ij} \) is a binary digram, then \( 1 \leq i < j \leq n \).

The binary digrams that have been described differ from conventional digram statistics. The probability of occurrence of each letter pair has been quantized into a 0 or 1, denoting whether there is a zero or nonzero probability of occurrence, respectively, of each letter pair. Generally this involves a loss of information. On the other hand, the necessary storage has been reduced to a fraction of the original requirements. The information in an entire binary digram can be stored in 26 data cells if each cell can store 26 bits or more, as opposed to the 676 cells previously necessary. The second new characteristic that has been introduced is the use of digrams for any two positions in a word. This differs from two approaches used in the past. The first is the standard method of using all contiguous pairs of letters, disregarding their position in the word, in the estimation of the digram probabilities. The second approach, used by Carlson [10], involved the use of position of contiguous pairs and
triplets for the calculation of position dependent digram and trigram probabilities. This concept has been generalized in this paper to encompass any pair of noncontiguous positions. Usually, a digram for the \((i, h)\)th position (i.e., positions \(i\) and \(h\) in the words) contains information not contained in digrams for the \((i, j)\)th and \((j, h)\)th positions.

The postprocessor will have available to it a certain subset of the possible binary digrams, and these constitute the block in Fig. 1 called dictionary syntax. Before looking up a word \(a_k a_{k+1} \cdots a_{k+r}\) in the dictionary, the postprocessor checks that for every \(d_{ij}\) available to it the \((k, k)\)th entry of \(d_{ij}\) is a 1. It is because most of the entries in the digrams are 0 that most words can be eliminated very quickly. Any word which is consistent with all the binary digrams available to the processor is called an admissible sequence, and every admissible sequence must be looked up in the dictionary. Given a sequence of substitution sets, every admissible sequence which is located in the dictionary is called an admissible d-sequence, and every word in the dictionary is called a d-sequence. Note that for a given sequence of substitution sets every admissible d-sequence is a d-sequence; the converse is not true since most d-sequences cannot be formed from any particular group of substitution sets and, therefore, do not appear in the set of admissible sequences.

A simple example of the use of binary digrams is given in Fig. 2 where the alphabet consists of four letters, the dictionary has 5 three-letter words, and substitution sets have two letters. Of the 8 three-letter sequences that can be formed, only two are admissible, and only one of these is in the dictionary.

The binary digrams just defined will consist mostly of zeros if the words in the dictionary are very similar in the positions for which the digrams are calculated. In general, the more zeros in a digram, the greater will be its ability to reject sequences. Therefore, as a rough guide, the more digrams used and the fewer ones they contain, the smaller will be the number of admissible sequences which require dictionary lookup. The rate at which incorrect words are produced by the recognizer and the rate at which words are rejected by the recognizer does not depend upon choice of digrams, but is highly dependent upon the dictionary. Clearly if there is never more than one admissible d-sequence then both the error rate and the rejection rate are zero, provided that every substitution set contains the correct character. Given a dictionary of \(N\) words it is possible to predict very accurately the behavior of the recognizer.

### ERROR AND REJECTION RATES

Let us assume that the correct letter is always in a substitution set, and for the purposes of this paper, let us further assume that the remaining letters in each substitution set are determined randomly and that the \(N\) words in the dictionary have \(a\) priori probabilities \(p_1, p_2, \cdots, p_N\). Thus, the correct word is always an admissible d-sequence, and we wish to determine the probability that no other admissible sequence is in the dictionary. Suppose in some run that there is a second admissible d-sequence which differs from the correct word by \(k\) letters. For example if horse is the correct word, house differs by one letter. We say that the second word differs from the correct word by a \(k\)-letter substitution, or in this example, by a one-letter substitution. Note that the summation of the \(k\)-letter substitutions, as \(k\) is varied, is equal to the number of admissible d-sequences minus one (since each \(k\)-letter substitution is an admissible d-sequence that may be confused with the correct word).

In order to determine, for a given \(k\), the probability that there is no \(k\)-letter substitution, an analysis of the dictionary is performed. Let \(w_j\) be the \(j\)th word in the dictionary, \(1 \leq j \leq N\), and let \(m_{jk}\) be the number of words in the dictionary differing from \(w_j\) by a \(k\)-letter substitution. Then the coefficient

\[
\gamma_k = \sum_{j=1}^{N} m_{jk} p_j
\]

denotes the expected value of the number of d-sequences differing from the correct word by a \(k\)-letter substitution. Now let \(l\) be the length of the words in the dictionary. There are \(\binom{l}{k}\) \(2^k\) ways in which a \(k\)-letter substitution, \(k \leq l\), may occur so that the probability of obtaining a d-sequence by substituting \(k\) arbitrary letters into \(k\) arbitrary positions of the correct word is given by

\[
\beta_k = \left(\frac{\gamma_k}{\binom{l}{k}}\right) 2^k.
\]

If the substitution sets all contain \(S\) letters there are

\[
\alpha_k = \binom{S}{k}(S - 1)^k
\]

ways of substituting \(k\) letters into \(k\) positions so that the probability of exactly \(r\) occurrences of a \(k\)-letter substitution in a given trial is

\[
p(k, r) = \binom{\alpha_k}{r} (\beta_k)^r (1 - \beta_k)^{\alpha_k - r}.
\]

Since the occurrence of a \(j\)-letter substitution and a \(k\)-letter substitution are mutually exclusive when \(j \neq k\), the probability of no substitutions occurring at all is given by

\[
P(0) = \prod_{k=1}^{l} p(k, 0)
\]

and the probability of exactly one substitution is given by

\[
P(1) = \sum_{k=1}^{l} p(k, 1) \prod_{j=1}^{l} p(j, 0).
\]
The probability of exactly \( t \) substitutions is calculated in a similar manner but is not given here. \( P(0) \) is the probability that the number of admissible \( d \)-sequences is exactly one, the correct input word, and we refer to \( P(0) \) as the recognition rate.

It is also of interest to know the functional dependence of \( P(0) \) on the dictionary size. Assume that the dictionary is enlarged by a factor of \( \delta \) with words having the same statistical properties. Each probability \( p_j \) decreases by a factor of \( (1/\delta) \), but each \( m_j \) increases by \( \delta \). Thus, \( r_j \), and hence \( \beta_k \), \( 1 \leq k \leq l \), grows linearly with \( N \). From (4) and (5) one obtains

\[
P(0) = \prod_{k=1}^{l} (1 - \beta_k)^{x_k}. \tag{7}
\]

Expanding each factor by the binomial expansion and neglecting large powers of \( \beta_k \), since \( \beta_k \) is extremely small, one obtains

\[
P(0) \simeq \prod_{k=1}^{l} (1 - \alpha_k \beta_k). \tag{8}
\]

If \( \alpha_k \beta_k \) are still relatively small, the polynomial for \( P(0) \) in (8) becomes

\[
P(0) \simeq 1 - \sum_{k=1}^{l} \alpha_k \beta_k \tag{9}
\]

so that when \( P(0) \) is close to 1 it depends linearly on \( N \). Finally, using (2) and (3),

\[
1 - P(0) \simeq \sum_{k=1}^{l} \alpha_k \beta_k = \sum_{k=1}^{l} \gamma_k \left( \frac{S - 1}{25} \right)^k. \tag{10}
\]

In other words, the percentage of trials which are expected to have more than one admissible \( d \)-sequence is a linear function of the size of the dictionary.

Finally, let us investigate the total number of occurrences of \( k \)-substitutions. The expected number of \( k \)-substitutions is given by

\[
e_k = p(k, 1) + 2p(k, 2) + 3p(k, 3) + \cdots. \tag{11}
\]

Thus from (4) and the fact that \( \beta_k \) and \( p(k, 2) \) are very small, the total number of \( k \)-substitutions is approximately

\[
\sum_{k=1}^{l} e_k \simeq \sum_{k=1}^{l} p(k, 1)
\]

\[
\simeq \sum_{k=1}^{l} \beta_k \alpha_k (1 - (\alpha_k - 1) \beta_k) \simeq \sum_{k=1}^{l} \beta_k \alpha_k \tag{12}
\]

which is the same as (10). This is to be expected, since if we assume \( p(k, 2) \) is small, whenever there is more than one admissible \( d \)-sequence, it is very likely that there are exactly two.

**Experimental Results**

The system that has been described has been simulated and tested in a number of experiments using randomly generated substitution sets and equal a priori probabilities \( p_1, \cdots, p_N \). The dictionary that was used consisted of 350 seven-letter words which were arbitrarily selected from common words in the English language. A sample of these words is given in Table I. If dictionaries for several different word lengths were used, the total dictionary size would be on the order of 1000 or 2000 words. Experiments were conducted to determine the average number of admissible sequences and admissible \( d \)-sequences, as well as error-rejection rates, as functions of the size of the substitution set and as functions of the size of the dictionary.

In all experiments all diagrams \( d_i \), such that \( j = i + 1 \) and \( j = i + 2 \), were used for the dictionary syntax. This means that for seven-letter words 11 digrams were used. In every experiment 5000 words were classified. Whenever an admissible \( d \)-sequence was found, it was determined in how many letters it differed from the correct word, and for each \( k \) the total number of \( k \)-substitutions were counted. Since \( p(k, r) \) is very small for \( r > 2 \), the expected number of \( k \)-substitutions in 5000 classifications is given approximately by

\[
r_k \approx 5000 \left( p(k, 1) + 2p(k, 2) \right). \tag{13}
\]

The values of \( r_k \) in (13) were compared with the number observed by a \( \chi^2 \) test, and the probabilities of observing higher values of \( \chi^2 \) are given on the graphs. \( \chi^2 \) was computed by placing \( r_1, \cdots, r_l \) into \( \lambda \leq l \) classes by lumping all \( r_k \)'s with values less than 5 into a single class. Then \( \chi^2 \) was computed according to (14), where \( \hat{c}_j \) and \( c_j \) are, respectively, the observed and expected frequencies for class \( j \).

\[
\chi^2 = \sum_{j=1}^{l} \frac{(c_j - \hat{c}_j)^2}{\hat{c}_j} \tag{14}
\]

with \( \lambda \) degrees of freedom. \( \chi^2 \) used in this way is sensitive to deviations from any of the individual \( r_k \)'s.

**Experiment 1:** Under the assumed characteristics of the modified recognizer, there will be at least one admissible \( d \)-sequence per classification. We refer to the number of admissible \( d \)-sequences minus the number of trials as the total number of substitutions. Each substitution is therefore a \( k \)-substitution for some \( k \). In Fig. 3 the total number of substitutions observed in 5000 classifications is plotted for dictionary sizes of 0 to 350 words and with the substitution set fixed at five letters. Recalling (12) and the fact that \( \beta_k \) depends linearly on \( N \), we expect the number of substitutions to depend almost linearly on dictionary size. Other than the single point at \( N = 350 \), the experimental results and the \( \chi^2 \) test based on (13) appear to verify this hypothesis.

<table>
<thead>
<tr>
<th>TABLE I</th>
<th>SAMPLE OF 50 SEVEN-LETTER WORDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>fashion</td>
<td>quarter</td>
</tr>
<tr>
<td>serious</td>
<td>stomach</td>
</tr>
<tr>
<td>blanket</td>
<td>sweater</td>
</tr>
<tr>
<td>morning</td>
<td>frantic</td>
</tr>
<tr>
<td>evening</td>
<td>speaker</td>
</tr>
<tr>
<td>studies</td>
<td>destiny</td>
</tr>
<tr>
<td>booklet</td>
<td>compile</td>
</tr>
<tr>
<td>measure</td>
<td>lasting</td>
</tr>
<tr>
<td>picture</td>
<td>courage</td>
</tr>
<tr>
<td>ceiling</td>
<td>nostril</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Fig. 3. Substitutions and admissible sequences versus $N$ with probabilities of higher $\chi^2$.

Fig. 4. Substitutions and admissible sequences versus $S$ with probabilities of higher $\chi^2$.

As an experimental measure of the computational complexity of the postprocessor, the average number of admissible sequences per classification is also plotted in Fig. 3 for a postprocessor which uses the 11 digrams previously described. This appears to be a reasonable measure since this is the average number of sequences that must be looked up in the dictionary for each word that is classified.

Experiment 2: In Fig. 4 the total number of substitutions observed in 5000 classifications is plotted for substitution sets of size varying from one to seven letters, with the dictionary size fixed at 300 words. The expected values are plotted and the $\chi^2$ test is again applied. Also, the average number of admissible sequences per classification is plotted.

Experiment 3: Although the average number of admissible sequences per word processed may be quite acceptable, in some instances a relatively large number of admissible sequences may appear which must be looked up in the dictionary. The peak in the discrete probability distribution for the number of admissible sequences is substantially below the mean, but the distribution has a long tail reaching relatively large values. The distribution in Fig. 5 has been estimated from 5000 classifications for $N=300$, $S=5$, and 11 digrams, and is typical in shape.

If we wish to bound the maximum number of dictionary lookups, we may choose to reject when the number of admissible sequences exceeds a threshold, $T$. Thus, as $T$ is lowered, the rejection rate increases and the error rate decreases. Fig. 6 is a graph of the error-rejection tradeoff for $N=300$ as $T$ is varied under the decision strategy that whenever the number of admissible sequences is below $T$, we accept the admissible $d$-sequence having the highest $a$ priori probability (in our case, where the probabilities are equal, we guess). The curves in Fig. 6 have been plotted from the experimental results rather than from computations. By noting the scales, one can see that a large increase in the rejection rate is necessary to obtain a noticeable decrease in the error rate. Table II lists the error and rejection rates for two different decision strategies. Note that the error rate for the second strategy is approximately one-half the rejection rate of the first strategy. This is due to the assumption of
A measure has been developed for the expected error and rejection rates, given the dictionary, as a function of the size of the substitution set of the general recognizer. Using 300 seven-letter words, an error rate as low as 1.81 percent, with 0 percent rejection, is achieved while the recognizer is allowed to output five alternatives for each character. This means a recognizer must be designed only to output the correct character as a member of a set of characters, whose size is approximately 1/5 of the alphabet. The authors strongly conjecture that, with a fixed dictionary size, recognition rates will increase substantially as the length of the words in the dictionary is increased. However, the recognition rate achievable in any particular circumstance is surely dependent upon the similarity of words in the dictionary, and the size of the substitution sets may have to be adjusted to achieve desired error rates. Nevertheless, even with $S$ equal to 2, the requirements on the character recognizer’s accuracy have been greatly relaxed.

The substitution sets in our work, other than the single correct character, were randomly generated, and it is true that it is somewhat unrealistic to assume a uniform distribution. The authors plan to obtain a better estimate of the recognition rate when the distribution of substitution sets is nonuniform by using an approach similar to the one presented in this paper. Also, knowledge of the $a$ priori dictionary word probabilities will be useful in improving performance.

It should be noted that the approach outlined by Bledsoe and Browning [2] involved the use of a substitution set equal in size to the entire alphabet but with probabilities associated with the letters. Rather than determine a confidence level for each possible string that can be formed from the substitution sets, the combinatorial explosion is avoided by determining this confidence level for each word in the dictionary. In our application, this might correspond to a postprocessor that checks each word in the dictionary, letter by letter, to see if it can be constructed from the substitution sets. The Appendix is a comparative analysis of the computational requirements of this alternative method with our contextual postprocessor. In all cases considered the postprocessor presented in this paper was more efficient; for $S=5$ and $N=300$ it was faster by a factor of about 3. Whereas the amount of computation required by the alternative approach is constant and fixed by the size of the dictionary, the number of computations in our algorithm can be reduced because it is a function of the amount of storage used for digrams, trigrams, etc. The recognition rate of the two methods would be exactly the same.

Many topics are open for further investigation. Although the lowest error and rejection rates that are achievable for a fixed $S$ depend upon the dictionary, the computational requirements for such a system are dependent upon the binary digrams and the specific implementation algorithm. It is still necessary to develop a measure for selecting the most useful subset of binary digrams and predicting the number of admissible sequences. The value of binary trigrams also should be determined.
APPENDIX

Computational Requirements

The following analysis is a comparison of the computation required by our contextual postprocessor with that of an alternative postprocessor that checks each dictionary word to see whether it can be constructed from the given substitution sets.

The alternative postprocessor will be discussed first. For any word in the dictionary it must be determined whether the first letter is in the first substitution set, the second letter in the second substitution set, etc. When any letter of a word does not appear in the corresponding substitution set, that word can be rejected. If the substitution set is of size \( S \), the probability of not finding the first letter among the \( S \) alternatives is \((26-S)/26\) with \( S \) symbol comparisons required. The probability of the first letter being found and the second letter not appearing is \((S/26) \times (26-S)/26\), requiring an average of \((S+1)/2+S\) symbol lookups. Extending this analysis, the expected number of comparisons necessary for a dictionary of \( N \) words each of length \( l \) is given by

\[
E_1 = N \left[ \sum_{i=0}^{l-1} \frac{(S/26)^i}{26-S} \left( \frac{i(S+1)}{2} + S \right) \right] + \frac{(S/26)}{2} \left( \frac{i(S+1)}{2} \right).
\]

Although \( E_1 \) is not an exact computation measure, the lower bound on the number of comparisons is given by \( E_2 \) where

\[
E_2 = (N-1)S + l.
\]

This bound is obtained by noting that the best the algorithm can do is reject every word but one in the first substitution set, while finding the proper word as the first character in each substitution set.

Suppose our postprocessor is implemented using the following (nonoptimal) algorithm, which requires 11 digrams for seven-letter words. A list of symbol pairs which is admissible by \( d_{12} \) is constructed from the first two substitution sets. Then, using this list and the third substitution set, a list of letter triplets, each of which also is admissible with \( d_{13} \) and \( d_{23} \), is constructed. The procedure is continued, and the sixth list is exactly the list of admissible sequences. Each of these must then be looked up in the dictionary. The number of symbol lookups required by this algorithm will be denoted by \( E_3 \).

Table III is a comparison of \( E_1 \), \( E_2 \), and \( E_3 \) as a function of \( S \) for a dictionary of 300 words. \( E_3 \) was determined experimentally due to the complexity of determining a good theoretical estimate. It was assumed that the equivalent of two symbol comparisons was required to look up each admissible sequence in the dictionary by a “hashing” technique; this number is added to the average number of digram references required by the algorithm. The postprocessor using digrams is distinctly more efficient at the cost of storage.

ACKNOWLEDGMENT

The authors wish to express their appreciation to their colleagues, Prof. C. C. Foster and Prof. J. W. Ulrich, for their helpful discussions during the course of this work.

REFERENCES


