THE USE OF CONTEXT IN CHARACTER RECOGNITION

Edward G. Fisher

COINS Technical Report 76-12

(July 1976)

Supported, in part, by:

Office of Naval Research Contract No. N00014-67-A-0230-0007

THE USE OF CONTEXT IN CHARACTER RECOGNITION

A Dissertation Presented

By

EDWARD G. FISHER

Submitted to the Graduate School of the
University of Massachusetts in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

March 1976

Computer Science

Supported, in part, by:
Office of Naval Research
Contract No. N00014-67-A-0230-0007

THE USE OF CONTEXT IN CHARACTER RECOGNITION

A Dissertation Presented

By

EDWARD G. FISHER

Approved as to style and content by:

[illegible]

Dr. Edward M. Riseman, Chairman of Committee

[illegible]

Dr. Michael A. Arbib, Member

[illegible]

Dr. Allen R. Hanson, Member

[illegible]

Dr. Conrad A. Wogrin, Member

[illegible]

Professor Robert M. Graham
Department Chairman
Computer Science

To Odette

ACKNOWLEDGEMENTS

The author wishes to thank Professor Edward M. Riseman for the assistance which was provided in writing this dissertation and for the assistantships which were provided while the author was at the University of Massachusetts. The author also wishes to thank the other members of his committee: Professors Michael A. Arbib, Allen R. Hanson, and Conrad A. Wogrin for the assistance provided in finishing this dissertation and their helpful comments and criticisms.

The author sincerely appreciates the support which was provided by Professor Wogrin and his staff at the University Computing Center.

The author also wishes to thank the many fellow graduate students and computing center users without whose problems, comments, criticisms, and never-ending questions this thesis might have been done much earlier but would not have been as much fun.

Last but never least, the author wishes to thank Mrs. Joanne L. Sauppe for the long hours and great effort which she invested in the typing of this thesis and Ms. Janet Turnbull for her work in setting up and typing some of the figures.

ABSTRACT

The Use of Context in Character Recognition

(March 1976)

Edward G. Fisher, B.S., University of New Hampshire
M.S., Ph.D., University of Massachusetts

Directed by: Professor Edward M. Riseman

In this dissertation new algorithms are developed for the application of context in character recognition. Improvements ti binary n-gram algorithms are proposed which facilitate the use of different types of n-grams for a collection of words of varying lengths. Algorithms which utilize this extended data base for the detection, location, and correction of insertion, deletion, split, and merger errors are presented. A primary feature of the new algorithms is that the n-grams are anchored to one or both ends of the word.

Experiments are performed which show that the algorithms are effective for all of the error types. For example, in a dictionary of 10000 words, without a priori knowledge of the types of errors being processed, the contextual postprocessor was able to correct 81% of single substitution errors, 57.7% of deletions, and 63.2% of double substitution errors. Numerous experiments were performed to examine the performance of the postprocessing algorithms as a function of dictionary size, as a function of word length, and as a function of the size of the contextual data base.

The application of these techniques to postal reading machines is explored in the latter part of this work. A brief exposition of the problem of mechanically sorting mail in the Post Office is provided. The algorithms are further developed to employ additional binary n-grams to correlate the redundant information in city-state-Zip Code addresses.

They place a hierarchical emphasis upon the postprocessed data in order that the output may be viewed in the context of its relevance to the type of routing being performed.

Simulating a classification system which has a character error rate of 3 to 4%, errors were produced in 43% of the city-state-Zip addresses. The postprocessor was able to sort 97.5% of the erred addresses, yielding a gross sort to the correct region of 98.9% with an error rate of .04%. The CPP was able to sort 97.2% of the addresses to the correct city.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

1.1. BACKGROUND

The recognition of text by machine has been studied for many years. Machine reading of text has long been viewed as a means for eliminating many of the most tedious jobs in our society, such as: computer input of a hundred million tax forms each year; sorting of billions of pieces of mail per year; processing of credit card transaction slips; and sorting all of the checks which must go through a few banks and clearing houses and be charged to the proper account. Checks can be read with fewer than one error per million because all of the printing is done in a single font with magnetic ink by carefully controlled machines. The numbers on credit slips can be read with some reliability because the character set was designed for optimum readability. Social Security forms which are typed by either typewriters of various fonts or computer printers can be read well enough that machines can read 26 of 100 million lines of data; the remaining lines are input to a machine by key operators. Machines have also been installed to sort mail; however, in order to restrict these machines to a one percent error rate, it is necessary to force them to reject twenty percent of the mail they process.

From each of these examples, it might be noted that the success of the recognition system is related to the amount of control over the characters being read. Various attempts at recognizing uncontrolled hand-printed and hand-written material have been much less successful. It is not surprising that hand-writing is by far the worse of the two problems.

Many researchers of optical character recognition systems have noted the results of two important experiments in human recognition of characters and text. Neisser and Weene (1960) discovered that humans are only able to correctly identify 96.8% of isolated hand-printed characters. Miller and Friedman (1957) discovered that humans have difficulty reading individual characters but are capable of reading most text properly because of the context of the material.

The goal of this research is to demonstrate that context can be successfully applied to a large, real-world problem such as a postal reading machine. To this end various new algorithms are presented which develop the methods of binary n-grams to detect and correct a wide range of different types of errors. The algorithms are then extended to include a correlative ability for relating state names and Zip Codes to city names for application to the Post Office sorting problem.

1.2. FOREWORD

Chapter II reviews some of the research in the field of pattern recognition. The emphasis of the review is upon the various methods which have been used to improve the performance of pattern recognition systems. Among the methods is the use of context; it is shown that context is the best and most consistently reliable method.

Chapter III reviews the methods of binary n-grams. The algorithms and capabilities of the systems developed in the prior literature are discussed. The chapter also cites many of the results of the prior research.

Chapter IV broadens the scope of the method of binary n-grams, both in breadth and depth. Additional algorithms are presented to facilitate the processing of words of different lengths. The methods are expanded to include a capability for detection and correction of additional types of errors.

Chapter V presents the results of several experiments which demonstrate the effectiveness of the algorithms developed in chapter IV. The experiments evaluate the methods for different sizes of dictionaries and different sizes of contextual data bases; they also demonstrate the relationship between the effectiveness of the postprocessing techniques and the length of the words being processed.

Chapter VI presents a brief discussion of the Post Office sorting problem. The methods of binary n-grams are extended to utilize the structure and redundancy of addresses and Zip Codes. Experimental results are presented which demonstrate the viability of the algorithms.

Chapter VII concludes the dissertation. The major innovations introduced in this dissertation are summarized. Some extensions of the algorithms are presented.

CHAPTER II

REVIEW OF THE LITERATURE

2.1. INTRODUCTION

This chapter briefly reviews the field of pattern recognition in general. Because of the failure of most pattern recognition systems to perform as well as humans would expect them to, most of the researchers have selected a particular component of a pattern recognition system and attempted to improve the system by improving the component. Thus our emphasis will be upon the various means which have been researched in order to improve contemporary pattern recognition systems. Two of the methods stand out as providing outstanding improvements: the placing of constraints upon the input pattern space and the use of context. It is shown that, given a problem domain which cannot be controlled, context is the best method for improving the performance of a pattern recognition system.

The latter part of this chapter surveys context in pattern recognition. It is shown that context may be applied in two ways: context may be incorporated into the classified, and context may be applied as a postprocessor. Several research papers are reviewed for each of these methods.

2.2. PATTERN RECOGNITION

There have been many studies of the various aspects of pattern recognition and, more particularly, character recognition. Most of

these projects are surveyed by Nagy (1968) and Kanal (1974). Harmon (1972) surveys the recognition of print and script.

The problem design of primary concern is presented in figure 2.1. First, features are extracted from patterns (digitized characters). The features are then processed statistically by a classifier and output as "text," possibly garbled. The other two components, the preprocessor and the postprocessor will be discussed later in this chapter.

The automated recognition of various types of patterns seems to be bounded by some generally accepted error rates. For example, an estimate computed by Neisser and Weene (1960) for the recognition of handprinted numerals and upper case English characters by <u>humans</u> appears to be accepted as a lower bound on machine recognition of this class of characters. (See, for example, Harmon, 1972.) Neisser and Weene achieved a pooled best guess error rate of 3.2% when certain reasonable decisions were made about what not to call an error, such as 0-O confusions.

Several different approaches to solving the character recognition problem have been taken, and may have improved the system performance of the character recognizer being simulated. Each approach has usually attacked some particular component of the system design of figure 2.1.

2.2.1. Improving the Feature Set

A system component which has frequently been studied is the feature selection algorithm. The related problem is usually phrases as: "Given a pool of N features, select the best set of n features ($n \leqslant N$) for recogniting some class of patterns." The fact that there are $\binom{N}{n}$

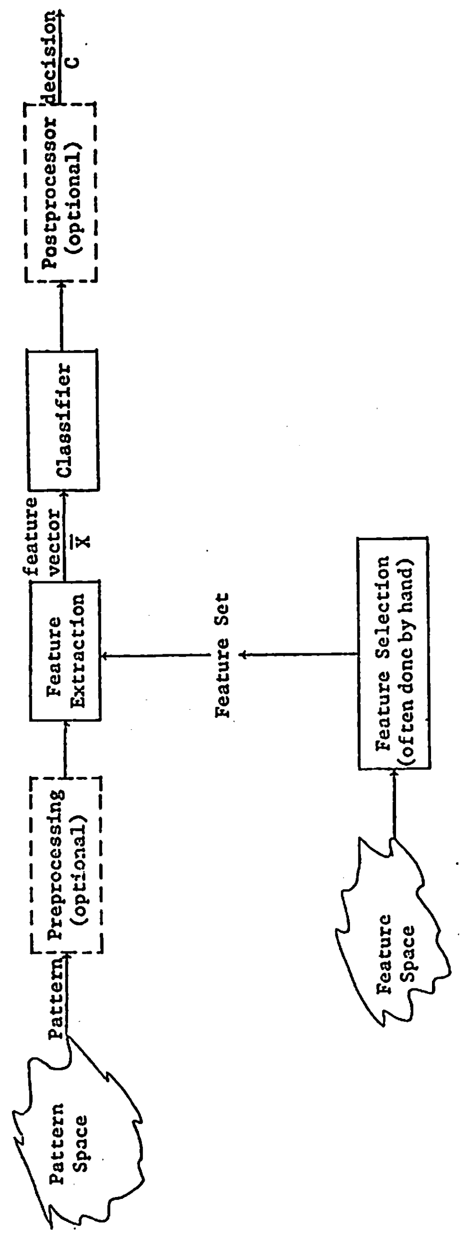

Figure 2.1 Schematic of the typical components of a pattern recognition system.

possible subsets makes the problem computationally difficult (e.g., $\binom{200}{30} = 4.1 \times 10^{35}$). Therefore, an algorithm which can select a "very good" subset is sought. Mucciardi and Gose (1971) experimented with several techniques for choosing subsets of pattern recognition properties in an EKG problem. Their weighted sum technique produced an error rate of 36% with 50 of 157 features. Fifty randomly selected features had an error rate of 42% and all of the other algorithms had error rates from 36 to 42%. (They cite other research as finding that disagreement among groups of electrocardiographers is as high as forty percent.) Their conclusions were not that they had found a best algorithm but that their own algorithm compared favorably with the others and that a choice would have to be made on the basis of implementation considerations and problem design.

Fisher, Hanson, and Riseman (1974) developed and compared several feature selection algorithms in a character recognition problem. The error rates of the algorithms they used ranged from 10.3% to 20.7% when 40 features were selected from a pool of 200. In this case it appears that choice of algorithm did make an appreciable difference although several of their algorithms did have error rates below 13%.

Another approach to feature set improvement is to select a larger set of features. Both Mucciardi and Gose and Fisher, et al., display results of such experiments. Mucciardi and Gose's experiments reduce the 36% rate for 50 features to 33% for 78 features. Fisher, et al., improve the 10.3% rate mentioned above for 40 features to 8.9% for 70 features. A phenomenon of this methodology is the fact that in some experiments the error rate has been known to increase when larger and larger feature sets were applied to a problem. This "peaking phenomenon" is discussed by Chandrasekaran and Jain (1975). There are several reasons for the peaking phenomenon: if the features are not independent, the covariance matrix becomes singular as soon as the feature set is larger than the training set; if the features are all independent, theoretical results indicate that an arbitrary increase in the size of the feature set will improve recognition only if the underlying model is Bayesian (the a priori distribution of the patterns is known). However, in practice it is not possible to obtain arbitrarily many independent features.

2.2.2. Improving the Pattern Space

If the feature set cannot be much improved perhaps the patterns themselves can. If the patterns can be enhanced, the quality of the measurements obtained should be better and the error rates lower. Two techniques for refining characters are image enhancement through preprocessing and application of constraints to the patterns in the pattern space.

Preprocessing as a means of refining images has occasionally been quite successful. Various preprocessing transformations are applied to the patterns in an effort to standardize them before feature measurement. The purpose of the transformations is to reduce the amount of variability in the patterns. Examples of such preprocessing are size normalization, noise extraction, skew normalization, and line thickness

standardization. The relative effectiveness of these schemes is dependent upon the sensitivity of the features to variation. For example, if the number of intersections through the middle of a pattern is a feature and the characters are allowed to move about in the pattern matrix, the feature is sensitive to such variation; however, if the character is centered in the pattern matrix, made to stand vertically, and noise is removed from the pattern, the feature should be more reliable. For example, Toussaint (1972) experimented with some position dependent features, e.g., masks which measured the density of fixed portions of the pattern matrix; application of size normalization to the input patterns reduced an error rate of 32.3% to 26.0%. Fisher (1974) applied size normalization to the same characters but in a problem which included many topological features (topological features are essentially position independent); error rate improvements were less dramatic, 10.9% to 9.1%.

Lin and Scully (1974) achieved an error rate of .4% on constrained hand-printed characters. Their problem design included the use of 49 character classes and required the subjects to print on a standard form. Each character stroke was to be on top of a line of the standard form. They state that the subjects who generated the data said that the constrained characters "were not overly complicated, were reasonably easy to get used to, and became progressively easier to use as the testing continued."

Research by others has occasionally indicated that people tend to dislike constraining their printing styles, become inefficient printers, and even sometimes rebel against the constraints. (Caskey and Coates, 1972; Devoe and Graham, 1968)

Troxel (1974) speaks of pattern recognition machines which make one error in every twenty to fifty thousand characters processed, provided that the source documents were generated by a particular type ball with a particular quality of ribbon and grade of paper. Thus, given a particular problem domain whose quality can be carefully controlled, some means exist to improve the performance of mechanical reading equipment by improving the quality of the source documents.

2.2.3. Improving the Classifier

There are many alternatives in the design of classifiers. Some of these pertain to the statistical assumptions made about the features, whether they are to be described by parametric, non-parametric, gaussian, linear, non-linear, or piecewise linear models. Other alternatives concern the decision mechanism: is it to be sequential or parallel? Further classifier design alternatives involve incorporating contextual information either in the classifier or in a special processor which is to operate on the text output by the classifier. Context will be discussed in greater detail in the next section of this chapter.

2.3. CONTEXT

If the problem is changed from that of character recognition to the problem of word or text recognition, the identity of characters can be considered in the context of those characters in the neighboring text. Shannon (1951) discusses the redundancy of English text and presents the

results of a few experiments which demonstrate the ability of humans to fill in gaps in text. Shannon estimates that English is so redundant that each character of English text actually conveys less than 1.3 bits of information (in contrast to $\log_2 26 = 4.67$ bits maximum for randomly occurring characters). The use of context is so interwoven with human recognition processes that we frequently recognize familiar objects and persons from a few contextual clues instead of looking for complete, detailed information. Humans read over typographical errors without noticing them. Detective novels often have mistaken identity as a key item of the plot. ("Well, the murderer and the defendant were both wearing a brown coat and a green hat!). We never question the meaning of the spoken word "bar" (bare or bear) because context always makes clear which of the two spellings and twenty or more meanings are intended by the speaker. When helping a child or adolescent with his reading, we are often asked, "What does this word mean?" And we often respond, "How is it used?" Context is a very important part of the human recognition process.

Miller and Friedman (1957) studied the ability of humans to correct a variety of mutilations in printed English texts. They found that the "average" person with limited time to work will not be able to correct passages perfectly if more than 10% of the characters are mutilated and that the job was even more difficult if the mutilations were random substitutions of erroneous characters. However, "superior" persons with unlimited time were able to correct text which had been 50% mutilated through deletion of either every other character or all vowels and spaces.

It is important to note that the use of context implies the recognition of aggregates of patterns. Because of this, system error and recognition rates are often given in terms of words instead of characters. The difference is significant because in the non-contextual (independent) recognition of patterns, a 10% character error rate on four-letter words translates directly to a 34.39% word error rate ($.3439=1-[1-.1]^4$). However, when context is used to recognize patterns errors are not independent; therefore, error rates (character vs. word) are not so clearly related; words with errors in them are more likely to have a second error than they are when errors are independent. Thus, there are slight differences; a few will be cited when available. However, just as most researchers have used data bases which don't permit comparisons of results, many have used only one means of presenting error rates.

2.3.1. Types of Context

There are several levels at which context may be applied in the recognition of patterns. These correspond to the various levels of detail being used to differentiate among the candidate patterns, whether they be characters, text, speech, or scenes

2.3.1.1. Global Contexts

At the highest level, there exist the cultural, temporal and

physical circumstances under which the text was generated. A system to use this sort of context would be at the forefront of research in artificial intelligence. Multiple levels of world knowledge provide various contexts in which to interpret the same data. In text recognition such a system would know that an Englishman would write "colour" instead of "color." In speech recognition or natural language processing the system would adapt itself to the vocabulary of, say, an eight-year old speaker or the accent of a West Virginian before attempting to resolve ambiguities. In scene analysis, the system would account for the season of the year and the angle of the sun while identifying the components of scenes. That is, such a system would use contextual information which is not immediately apparent and would continually adapt its contextual base to changes in the pattern space. Of course there are reasons for the fact that these techniques have not been implemented: they are very complicated and the lower level systems are not yet very well developed. For example, until there are good speech recognition systems, the refinement of adaptation to provincial pronounciation is an unnecessary embellishment.

2.3.1.2. Syntax and Semantics

The semantic and syntactic contexts in which words exist have been considered occasionally. Szanser's General Content check (1971a) is one example of a successful application of global content analysis in the recognition of text. The occurrences of "content" words are counted when they are encountered in a body of text. In a second pass words which have multiple resolutions are resolved on the basis of the most frequently occurring alternative of each of the ambiguous forms. Szanser also used some linguistic syntax rules to choose among multiple resolutions (1971b); this was primarily useful in the resolution of short words.

An interesting application of syntax and semantics to the automatic correction of spelling errors is in the restricted domain of programming languages. Duda and Hart (1968) researched a character recognition problem in which they attempted to "read" a FORTRAN program from coding forms. Their algorithms reduced a 9.3% character error rate to 2.4%, and they suggest that additional programming should have reduced it to 1.2%. They add, however, that the remaining errors would have eluded any additional clever programming.

Heinselman (1972) built a compiler addendum which searched for and found many spelling errors in student programs. Morgan (1970) describes several specialized techniques for efficiently incorporating spelling correction algorithms into compilers and operating systems. CORC and CUPL (Conway, et al., 1963a, 1963b; Walker, 1967) were two student oriented prog amming languages at Cornell University designed to find and correct spelling errors.

Alter (1968) applied contextual constraints to the automatic recognition of a "spoken FORTRAN" language. Only prelirinary results are presented in the paper.

2.3.1.3. Context among the Patterns

Context which has been used much more frequently has involved the statistical or linguistic relationships of letters within words. The work in this area has taken two paths: many researchers have designed classifiers which used contextual statistics within the classification process itself; and others have designed a subsequent machine, a postprocessor, which observes mutilated text and detects and corrects errors in the text. The postprocessor has also been designed as an independent data entry aid which was independent of any character recognition scheme per se but which still had the task of scanning text and detecting and correcting errors.

The two alternative methods are detailed in the next subsection of this chapter and in the forthcoming chapters. The first, context within the classifier, has involved various sequential decision techniques in which the characters are viewed in the context of their immediate neighbors. The second, the postprocessing technique, has sometimes taken a slightly broader view in which characters are viewed in the context of the entire word which contains them.

2.3.1.4. Context within the Patterns

A fourth area in which context can be used is within the structural context of the patterns themselves; i.e., the graphical, spatial, syntactic manner in which the components of the language elements are assembled. To use this context a pattern recognizer concerns itself with the characteristics of the parts of the character. Many researchers have concerned themselves with the stroke order of handwritten letters. (Ehrich and Koehler, 1975; Frishkopf and Harmon, 1961; Mermelstein and Eden, 1964; Sayre, 1973) Gold's Morse Automatic Decoder (MAUDE) (1959) was a language independent pattern recognizer which used rules for distinguishing between short marks and long marks (dots and dashes) and among symbol spaces, letter spaces, and word spaces, given the context of the surrounding marks and spaces. The relative duration of the marks and spaces of hand-sent Morse code varies among senders and even over the length of time of a single transmission by a single operator. The rules are therefore specific to the local context of structure. For example, the first rule is, "The longest of six successive spaces is almost always a long space." (The particular type of long space is determined by application of other rules at a later time.) While the character error rate was one percent over the several tested operators and several messages, 15 of the 184 test messages had more than 10% errors. Space errors (letter vs. word spaces) posed a serious problem; 60 of the messages had more then 20% space errors.

Cox, et al., (1974) considered the problem of font design with the hope that a pattern recognizer's knowledge of particular font characteristics would improve its recognition rates since documents are usually printed with a single font. This is not always true since Hennis (1968, p348) says that his group encountered many documents that were typed in different fonts and that they found at least one occurrence of five fonts on the same document. Shillman (1974) conducted

experiments to increase the understanding of human perception of differences among characters given the graphical context of the printing style. Fu (1974) discusses many of the methods of syntactic pattern recognition and provides an extensive list of references to the literature in this field.

2.3.1.5. Total Integration of Context and Pattern Recognition

At every level of automating the recognition of patterns or groups of patterns or understanding the words formed, context can be usefully employed. When putting the strokes together to form characters, a feature extractor can use already deduced information about the other strokes in the character. When deciding the identities of characters, a recognition system can use the information that it thinks is correct about the surrounding characters to make some final decisions. To further resolve ambiguities or understand the text they process, they can assemble great quantities of detail from the substance of the text and use this to further process the data as might a literary critic.

2.3.2. Context in the Classifier

Abend (1968) is a good reference on compound decision procedures of the sort used in this area. He shows that compound decision theory provides the optimal sequential procedure for taking context into account in the classification problem. The ba[illegible] for the research is that Bayes classification is known to provide the optimal classification scheme given the statistical constraints of the model; however prior Bayesian work ignores the context of the pattern. The prior probabilities P(C) of Bayes' rule

$$P(C|X) = \frac{P(X|C)}{P(X)} \quad P(C)$$

are replaced by transition probabilities to compute the maximum a posteriori likelihood for first order Markov-chain dependence

$$P(C_n|X_n, X_{n-1}) = \frac{P(X_n|C_n)}{P(X_n)} \sum_{C_{n-1}} P(C_n|C_{n-1}) \; P(X_{n-1}, C_{n-1}) \qquad (2.1)$$

where C is an arbitrary character label, C_j is an arbitrary label for the character in the j-th position, X is a pattern vector, x_j is the vector of the pattern in the j-th position, and the sum is over all possible values of C_{n-1}. (Abend credits himself and Raviv (1968) with independent discovery of this rule.) This formulation is the optimal sequential model; the decision on C_n is conditioned only upon its pattern vector X_n and a function of the pattern X_{n-1} and not upon any decision which might have been made about the identity of X_{n-1}. Abend also presents the suboptimal rule which reduces to the a posteriori likelihood

$$P(C_n|X_n, D_{n-1}) = \frac{P(X_n|C_n)}{P(X_n)} \quad P(C_n|D_{n-1}) \qquad (2.2)$$

where D_{n-1} is the decision made on the preceding character. In his analysis of error bounds on these two formulations, he found that the suboptimal procedure was almost as good as the optimal procedure when the pattern recognizer is good; however it is very poor when the pattern

recognizer cannot distinguish patterns well. If the pattern recognizer is good, $P(C_{n-1}|X_{n-1})$ will be near one for the correct value of C_{n-1} and near zero for all others; thus, the term $P(X_{n-1}, C_{n-1})$ will be largest for the correct decision and the summation in (2.1) will approach $P(C_n|D_{n-1})$.

Raviv (1967) simultaneously and independently developed the two above formulas; he also extended the formulations to a "look ahead" mode of decision and applied the decision rules in a pattern recognition problem. Raviv considers the pattern X_n in the context of the preceding and succeeding characters in both the optimal and suboptimal decision rules; the suboptimal decision rule in "look ahead" mode is

$$P(C_n|D_{n-1}, X_n, D_{n+1}) = \frac{P(X_n|C_n)}{P(X_n)} \quad P(C_n|D_{n-1}) \;\; P(D_{n+1}|C_n) \qquad (2.3)$$

Obviously the decision D_{n-1} can be arrived at in the same way as the decision on C_n; however Raviv leaves one to wonder at the means used to determine D_{n+1} since somehow something has to wait for the decision on C_n. Presumably some tentative decision is made about its identity for the purpose of identifying C_n, then the decision is discarded and the maximum a posteriori decision is computed as above for C_{n+1}.

Raviv performed many experiments, varying both the contextual data base and the pattern data base. In the first and one of his most successful experiments, the contextual data base was eight million characters of legal text, the pattern design set was 89076 samples of 53 characters from 28 different fonts of type. The data set used for testing the design was 27000 characters of legal text typed with an IBM Selectric with an elite font. Allowing for no rejections, 98% of the characters were recognized correctly when no a priori distribution information was used in the classifier; 99.3 with only the suboptimal formulation which did not use "look ahead" information; and 99.4% when the "look ahead" mode was used. The reduction in error rate from 2% to .7% and .6% is a significant reduction; it is proportionally the best which has been achieved with compound sequential decisions.

Raviv also experimented with a design factor to attempt to determine a relative weighting of contextual and measurement information. To employ the design factor an exponent F was applied to the feature measurement term, $P(X_n|C_n)$, in (2.3); a value of F=1.0 would be equivalent to not using a design factor and a smaller value of F would have the effect of placing less reliance upon the measurement information and more upon the contextual information. The experiments sought the optimal design factor. We cite the results of two more of his experiments. The same pattern set as described above was used (28 fonts). The contextual data base was 100 typical business documents. In "look ahead" mode and with no design factor (F=1.0), the machine recognized 96.3% of the characters; the optimal design factor (F=0.5) yielded a recognition rate of 96.6%. From these and other experiments, Raviv concludes that it is important to give the proper weighting to the input to be obtained from context. We feel that the difference is inadequate for so strong a conclusion and that the marginally better results obtained from the use of a design factor may be an artifact of the data. The use of the

suboptimal decision rule should not have adversely affected his results since he seems to have very good classification results; therefore the better results obtained from the use of a design factor are probably a measure of the lack of relationship between his test data and his design data. Also, the design factors were chosen as the result of several experiments on the test data; they were not again tested on a new test data set.

From other experiments Raviv concludes that use of bidirectional first order Markov statistics in the "look ahead" mode (2.3) is better than the use of the second order probabilities obtained by replacing $P(C_n|D_{n-1})\ P(C_n|D_{n+1})$ by $P(C_n|D_{n-1},D_{n-2})$ in (2.2) above. He also states that other preliminary results indicate that more specialized learning of the material to be processed will provide better results; however he also warns that such training restricts the data base and renders it less reliable than the more global training procedure used in the above experiments. We would agree with this and suggest that the specialized training might render the design factor to be meaningless.

Forney (1973) surveys various uses of the Viterbi algorithm and sketches an implementation of the Viterbi algorithm as applied to text recognition. The Viterbi algorithm is a recursive optimal solution to the problem of estimating the state sequence of a discrete-time finite-state Markov process observed in memoryless noise. Neuhoff (1975) used the Viterbi algorithm in conjunction with Bayes classification to recognize text. Neuhoff simulated a classifier with the confusion matrices published by Raviv (1967). Results obtained using diagram statistics were similar to those of Raviv (1967) although Neuhoff conjectures that the results whould have been better if he had used classifier likelihoods instead of the decisions of Raviv's paper. Forney (1973, p275) clains that the Viterbi algorithm is simpler to implement; however, since the Viterbi algorithm is essentially an algorithm for finding the least cost path through a graph, a careful evaluation of the two methods remains to be performed.

Donaldson and Toussaint (1970) and Toussaint and Donaldson (1972) investigated some simple contextual decoding algorithms to improve recognition system performance and/or reduce system cost. Application of their algorithms did reduce the character error rates from 19% to 14%.

Shimura (1973) describes recognizing machines with parametric and non-parametric learning methods which use contextual information. While this is, on the whole, a good paper, Shimura clouded the whole issue of the use of contextual information in his second paragraph where he states (p149-150):

> "...It is noted that if information about the preceding letter is used instead of the succeeding letter, recognition may be made based on wrong information when the preceding letter is misclassified. Thus we believe that the method the succeeding pattern is more desirable than that using the preceding pattern."

He is apparently unaware of Abend's work and interprets the use of contextual statistics as requiring that in the case of the preceding letter he use the suboptimal formula (2.2). However, in the case of succeeding characters he chooses the character C_n which maximizes

$$\max_{C_{n+1}} P(C_n,C_{n+1}|X_n,X_{n+1}) = \max_{C_{n+1}}(P(C_n,X_n)\ P(X_{n+1}|C_{n+1})\ P(C_{n+1}|C_n)) .$$

The letters C_n and C_{n+1} which maximize $P(C_n,C_{n+1}|X_n,X_{n+1})$ are the Bayes' (optimum) decisions for the two positions, given only the information about the two patterns; however, the decision C_n is made on this basis and X_{n+1} is reprocessed with X_{n+2} to make the decision C_{n+1}, etc. Although there is some sense of making an optimal decision here, it is neither optimal in the sequential sense of Abend nor in the recursive optimal sense of the Viterbi algorithm. Thus the question of which decision mode is more desirable remains unanswered and Shimura's contribution might be greater if he had supported his beliefs with data and done a better review of the literature.

Casey and Nagy (1968) report on a project which was primarily concerned with the clustering of patterns. Training on a set of 3000 characters, 26 unlabelled clusters were formed using a complicated clustering algorithm; then, a transition matrix was compiled. (At this point the cluster members are only labelled by numbers.) The clusters were then identified by an algorithm which matched the transition matrix to a digram frequency matrix for English in a manner similar to the solution techniques used for simple substitution cryptograms. Final error rates on 30000 characters of legal text are 0.2% for the "Prestige Elite" font of type and 2.0% for a script font.

For the special case where the context is generated by a two-state stationary Markov chain, Chu (1971) obtained an upper bound on the average error probability of an optimal recognition procedure based on compound decision functions. Chu admits that the restriction to two states is a drawback. However, he does claim that his results support the prevailing feeling that context cannot materially improve an otherwise unreliable recognition system and that the more interdependent the consecutive characters are, the more helpful is the use of context. It would be useful to know how unreliable a recognition system would have to be before such an application becomes useless. Further elaboration of the formulation may be found in Toussaint (1972) and Chu (1972).

2.3.3. Context in a Postprocessor

There have been many studies of the use of context in a postprocessor or as a data entry aid. Here it is the primary purpose of the processor to detect and correct errors in the text so that the corrected text can be either stored in a retrieval system or used as a key to retrieve data from a file repository. In the systems which have been implemented, the processed text is either used as a data retrieval key or output in manuscript form. As a data retrieval key, the text must be used to find other data in a data base; for this to be successful, accuracy of the data key is essential. Frequently this data is entered by a person at a teletype and, therefore, the spelling of the input datum is not assured. If the data is to be output in manuscript form, the originating agency would be as particular about the accuracy of the manuscript as if there were a person doing the tedious transcriptions. Szanser's system (1971a-b) was designed to output both court and parliamentary transcripts.

Many of the techniques used in these processors involved use of the dictionary of interest. There is no system which is more accurate for detecting errors than searching a dictionary, unless syntax or semantics is used. However, if the dictionary is large, correcting errors can require a great deal of computation unless either an associative memory or sophisticated algorithms are available. These algorithms generally select a subdictionary for searching; that is, they point to a small set of dictionary words which might contain the correction. Assuming that the error is of some prespecified type, if it can be shown that the correct subdictionary is always selected, then the algorithm has the same power as any full dictionary search algorithm.

As a simple example of subdictionary selection, suppose we are given a string of characters $X=x_1x_2\ldots x_m$, such that x_i is incorrect; and there exists y such that $Y=x_1\ldots x_{i-1}\ y\ x_{i+1}\ldots x_m$ is in the dictionary. An algorithm which selects the set of all m-letter words from the dictionary is a "correct" subdictionary selection algorithm. However, since the subdictionary of m-letter words has length proportional to the size of the entire dictionary, search time grows with the size of the dictionary. For a dictionary of length n, this algorithm could be considered to be of order $O(c_m \cdot n)$, $c_m \leq 1$; that is, the amount of computation required is linearly proportional to the dictionary size and the constant of proportionality is c_m, the expected proportion of words of length m in the dictionary. (Approximate values of c_m can be obtained from Kučera and Francis (1967).) Algorithms of order less than O(n) are desired; however, the algorithms in this field have not been studied for their timing characteristics.

The research in this area could be classified by several criteria. The most important criterion is the particular algorithm, if any, used to select a subdictionary since its effectiveness is, ultimately, the measure by which the algorithms should be compared. A second criterion is the size of the dictionary for which the algorithm is intended; an algorithm which is reasonably effective on a one hundred word dictionary might fall apart when given a one thousand word dictionary.

Since this processor is sort of an after-the-error-has-been-made processor, the third criterion is the type of errors which the algorithm is designed to detect and correct. The most obvious error when a pattern recognizer is involved is the substitution error; i.e., the substitution of one letter for another. Also of the one error per word variety are insertions and deletions. These are essentially spelling errors in which the author has either added or deleted a letter. The transposition error is frequently considered to be a single error. Of course, multiple combinations of all of these are possible.

Some early, theoretical discussions of the problems are presented by Glantz (1957) and Thorelli (1962). Glantz considered the problem only in terms of exact and near matches. He proposed comparing the input word to each dictionary word and computing a simple figure of merit as the ratio of the number of matched characters to the length of the longer of the two words; he would then select as a correction the

particular word which had the highest figure of merit. Thorelli describes the problem as being the study of two processors. The first is an error generator; the second is an error corrector, a degarbler. It is the task of the second processor to degarble the output of the first using the expectations of the original inputs and its knowledge of the process by which the first processor generates errors. The elegant part of the characterization is that it does not require that the first processor be a machine, only that there exist a model of its error generation processes and its inputs.

2.3.3.1. Abbreviation Techniques

Several researchers have investigated various abbreviation techniques: given an input word X, systematically throw away a number of letters, and then look up the abbreviated form in a dictionary of similarly abbreviated words.

Blair (1960), Davidson (1962), and Jackson (1967) developed systems which used various abbreviation techniques. Blair and Davidson assign a utility measure to each of the characters in the input word, depending on the particular character and/or its position in the word. The word is then abbreviated to four characters by discarding the characters with the lowest utility measures. Both methods work well on small dictionaries and, because of the abbreviation techniques used, are capable of working in the presence of the most likely of the substitution, insertion, and deletion errors. For example, from Davidson's paper, after keeping the first letter the characters most likely to be discarded are: vowels; H, W, and Y; and all but one occurrence of adjacent repeated consonants (adjacent after deletion so that the second "n" of the syllable "mem" would be deleted). Damerau (1964) provides a comparison of his method with that of Blair. Jackson uses a "reasonable" operator keyed abbreviation. In his system, nearly any reasonable abbreviation results in fast, accurate accessing of data from his data base of corporation names and aliases.

Tanaka, et al., (1971) designed a network which would output the correct word if the input word contained a minimal set of key letters in the proper positions. A key letter set, KL, is a set of letters in a particular set of positions which uniquely identify a word from among the set of dictionary words. G*RL is a KL of GIRL. The network for four-letter words would consist of 4x26 sets of switches labelled: 1A, 1B, ..., 1Z, 2A, ..., 4Z. Given the three-word dictionary, "GIRD," "GIRL," and "SUIT," the KL's of "GIRL" are "G*RL," "G**L," "*IRL," "GI*L," and "*IRL"; "G**L" is represented in the network as a circuit which connects switches serially at 1G and 4L. When both switches are closed the circuit is able to "respond," representing the fact that the KL belongs to the word "GIRL." Inputting the one-error string "GURL" to the network would close gate 2U for "SUIT," gates 1G and 4L for "GIRL," and gates 1G, 3R, and 4L for "GIRL." The network correction technique requires that the number of KL responses for candidate words meet a minimum threshold N; in this example, N=1 would cause rejection of the word because there are two qualifying responders, but N=2 would correct the word since only "GIRL" responds twice. Using a dictionary of 124

seven-letter words, setting N=5, using KL's of three letters or fewer, and simulating a classifier error rate of 10% (leading to a word recognition rate of 48.39%), the network improved recognition of words to 95.33% with an error rate of .16% and an undecided rate of 4.52%. The power of the technique remains to be demonstrated since their dictionary was not very large.

Tanaka and Kasai (1972) developed a method called Ordered Key Letters (OKL) which generalized upon the earlier KL method to include correction of insertion and deletion errors. Thus, the precise position of a letter was no longer important as long as the key letters maintained their precise relative order. After modifying the OKL technique to use a uniform length OKL (MOKL), they performed a number of experiments. With a dictionary of 610 five to nine letter words and MOKL's of four letters, the MOKL system was able to correctly identify 94% of single error seven-letter words and 67% of double error seven-letter words.

The key letter method is effective for substitution errors while the ordered key letter method is effective for substitutions, insertions and deletions. Tanaka and Kasai (1972) compared their method to the earlier KL method as well as a minimum distance method and the methods of Blair and Vossler (to be described later). Blair's method did not compare favorably to MOKL because Blair used abbreviation methods which ignored the actual information content of the letters in the words in the dictionary. Vossler's method had a 90 to 95% correction rate--though it is difficult to compare the methods on this basis alone.

2.3.3.2. Dictionary Methods

Bledsoe and Browning (1959) provided the first quantitative evaluation of context in a pattern recognition system. The process was that the classifier would compile all of the $P(C_n|X_n)$ probabilities and then, using each dictionary word of the correct length, compute a figure of merit for the word using those probabilities; the dictionary word with the highest figure of merit would be chosen as the word to be output. Though the procedure is unsatisfactory computationally, the paper did show the importance of context. Applied to cursive script, the methods successfully read 94.32% of the data; the next best method read only 60.% correctly.

In his description of the IBM 1975 Optical Page Reader, Hennis (1968) proposed an ingenious procedure which was not implemented but which clearly illustrates the relative values of correct recognitions versus errors and rejections. Proposing the use of a million word dictionary of American surnames, he suggests that each name either be accepted as read, changed and then accepted, or changed and then rejected. Thus, "Jones" is to be accepted, but "Janes" is to be changed to "Jones" and then accepted because the recognition process is likely to erroneously call a "Jones" a "Janes" fifty times for every two "Janes's" that it is likely to encounter; thus, accepting "Janes" as "Jones" is expected to fix 25 times as many errors as it should cause. On the other hand, "Bones" might be converted to "Jones" and then rejected

because the probability of error of such a change is greater than 5%. This technique can correct any likely error simply by placing the appropriate misspelling in the dictionary. Any form, whether correct or incorrect, which does not occur in the dictionary may be rejected since it does not occur often enough to have a significant effect on the system reject rate. Hennis did overlook the fact that adding the most likely erred forms of names to his list of American surnames would greatly enlarge the storage needed; for example, including several variants of "Smith" and only a couple variants of the least common names like, say "Pietrowski" would expand the dictionary significantly.

Szanser (1968, 1970a-b, 1971a-b) developed a technique called elastic matching which is capable of correcting all single errors except transpositions. Each dictionary word is coded into a series of computer words as bit maps. As described in the 1969 paper, words are broken into parts (lines) and then encoded into a series of computer words, one computer word per line. A line is a sequence of contiguous, alphabetically ordered letters from a word. For example, "BLOUSES" is broken up as "BLOU/S/ES." The line "BLOU" is encoded in a computer word as 1's in the second, twelfth, fifteenth, and twenty-first bits, for "B," "L," "O," and "U," respectively. A matching operation (exclusive-or'ing of lines) against "BLOV/S/ES" would produce a two-bit discrepancy in the first line, signifying a possible substitution error. Other substitution errors, such as "BLOUZ/ES" and "BLO/NS/ES" are also recognized by their particular mismatching characteristics, as are deletion and insertion errors, such as "BLOS/ES" and "BLOU/S/S/ES." An input word with, say, r lines and m letters, is matched against all of the subdictionaries whose words are described by the (line,letter) pairs (r,m), (r-1,m), (r-1,m-1), (r,m-1), (r+1,m), (r+1,m+1), and (r,m+1). If no near match is found, the word is either not in the dictionary or does not have the right type of error. If precisely one discrepancy is found; i.e., one bit is different in the coded form, a candidate for the correction has been found. A few search algorithm improvements have been presented (Szanser, 1971b) which improve the search time by factors of 4 and 16. (In absolute terms this represents reducing 3.3 to .9 and 16.0 to 1.0 seconds of CPU time on a KDF9 computer, but it is difficult to meaningfully compare timing figures among machines.) Szanser (1971a) later permutes the letters to be included in a line to a non-alphabetic ordering; the reason for the new ordering is that it results in a smaller average number of lines per dictionary word and, therefore, lowers the storage requirement of the dictionary.

Price (1971) describes the organization of a 71000 word dictionary into a search tree for a Palantype transcription system. A Palantype machine is a mechanical shorthand machine (typewriter) used in England; syllables are recorded as separate "chords" on separate lines of a strip of paper. Each node of the dictionary tree represents a chord or subtree of words similar to a parse tree of a grammar. He describes search algorithms as well as a number of dictionary modifications which had to be made because of Palantype ambiguities such as "fourth," "forth," and

"for the" and "we" and "wee." The problem was designed so that the solution would include an editor who could review the computer transcriptions and resolve the labelled ambiguities instead of having each Palantypist transcribe her own Palantype tape. Though his project met its fundamental goals, Price admits of two shortcomings: operator performance was not of a sufficiently high standard for the system to work properly and Palantype transcription itself was being abandoned in favor of audio tape transcription. Szanser's elastic matching algorithms (1971a-b) were designed to work with Price's system but with operation times of about one second per word for correcting errors, it was hardly satisfactory.

Giangardella, et al., (1967) developed a vector representation for words. Any two of the three components, magnitude, angle, and distance, are sufficient to identify a word. Words which contain the same letters ("bear" and "bare" and "polo" and "pool") have the same magnitude but the angles are different if the spellings are different; if two words have the same magnitudes but different angles the system checks for a transposition error. If the input word has a substitution, insertion, or deletion error, its magnitude cannot differ from that of the correct spelling by more than a fixed amount; thus the system selects from the dictionary all words whose magnitudes are within a specified range. Of 210 common misspellings of a set of words, 13 were uncorrectable because they were the correct spellings of other words in the 4286 word dictionary. Another 14 were discarded because "they contained more than one type of spelling error." The remaining 183 words which were deemed correctable contained 11 transpositions, 64 substitutions, 78 deletions, and 30 insertions. There are several flaws in the methodology of this paper. The 14 errors which were discarded were rejected by hand; Giangardella could have subjected them to machine processing so that the reader could see whether the system would reject them or would improperly modify them while attempting to correct them. Further, it is said that they were rejected because they contained "more than one type of error"; while this is sufficient cause for rejection, the less stringent criterion of "more than one error" is all that is necessary for rejection. Finally, Giangardella does not provide any information about the system's ability to correct the 183 errors; he only cites the fact that it corrected 100% of 37 other errors which had been artificially generated in 15 apparently unrelated words.

Damerau (1964) coded words into strings of 28 bits: one per alphabetic character, one for digits, and one for special characters. A bit in the code word is one if the corresponding character occurs in the word; duplicated characters are not counted again so that "BLOUSE" and "BLOUSES" have the same codewords. When a word is input to the processor it is coded and the codeword is compared against all dictionary codewords. If there is a match the input word is compared character by character to the corresponding dictionary word. If no successful matchings occur, the word is not in the dictionary and a second pass is begun to correct the word. The codeword is exclusive-or'ed with each of

the dictionary codewords; if the result has two or fewer bits that are one, it is possible that the input word is the result of applying a substitution, deletion, insertion, or transposition error to the dictionary word so the word is then compared to the dictionary word to check for such a possibility. Damerau tested his method with a dictionary of 1593 (?) words: it correctly identified 96.4% of 964 [sic] words with spelling errors. He tested Blair's method on his problem; Blair's algorithm correctly identified 81% of the words in error. Damerau does point out, however, that many of the 964 errors are machine errors; as we mentioned earlier Blair's method is particularly geared to correction of the most common of human errors so this comparison of methods might be considered irrelevant.

Thorelli (1962) proposed that a measure of similarity be compiled between a word to be processed and an appropriate dictionary subset (selected by length). He admitted the difficulty of compiling many of the transition probabilities necessary for his measure of similarity; transition in this case refers to the garbling process rather than letter adjacencies. Okuda, et al., (1975) formalize this with a Weighted Levenshtein Distance (WLD) as the measure of similarity. Computation of the WLD requires dynamic programming to find the least-cost path through a graph. The weights of the WLD are the relative costs of substitutions, insertions, and deletions. These weights are assigned to the edges of the graph with zero as the cost of a letter's being substituted for itself. The method is capable of correcting substitutions, insertions, and deletions.

Okuda and Tanaka propose a hardware realization of their method in which each dictionary word is represented by a network. When a word is input to the processor, it is transmitted to each network in the dictionary. If the input word has m letters and a dictionary word has n letters, the corresponding network has nm comparators to establish the corresponding substitution-cost edges. The costs of the transitions are implemented as delays: the delays which represent insertions and deletions are preset; the delays which represent substitution costs are set by the comparators at either zero or a predetermined cost. The WLD is computed as the number of clock pulses required to get a signal from the initial node N_{00} to the terminal node N_{nm}; the signal traveling the least-cost path is delayed less than all other signals. The quickest word to respond in the dictionary is the one with the least WLD. While this could certainly be implemented with LSI circuitry, it is quite complicated. Assuming the use of a standard chip for, say, a maximum word length of 12 letters, the comparators alone would require more than 2000 gates: 144 five-input and-gates, 720 two-input or-gates, and 1440 two-input and-gates, ignoring the fact that LSI circuitry only uses nor-gates or nand-gates. It would also require 100 three-input or-gates for the nodes of the network and 363 delay units for the edges with at least seven gates per delay unit plus whatever number of gates is necessary to count the delays. Together with driving circuits, input and output logic, etc., they are proposing at least 5000 gates per chip. The chip design is feasible. However, how much power is required to run, say, a

five thousand word dictionary or even to boradcast a single word to all of the networks in the dictionary? Further, what is to be done if the delay logic is kept simple and new chips have to be built to set new delay values? It is a good design but it is also one which will require careful cost analysis before implementation.

Alberga (1967) reported on several algorithms for computing measures of string similarity.

2.3.2.3. <u>n-Grams</u>

A number of algorithms have been developed which used various sorts of n-grams. A probability digram is a matrix whose elements d_{ij} represent the likelihood of the letter pair C_iC_j occurring in adjacent positions in any word in the dictionary. This notation is immediately extensible to trigrams and quadgrams which require commensurately more storage; for an alphabet of size d, an n-gram requires d^n storage locations ($(d+1)^n$ if <u>space</u> is an admissible character).

Sitar (1961) used probability digrams and trigrams to detect and correct substitution errors and to correct rejection errors. The algorithm which Sitar used to locate a substitution error with a probability digram would look up the probabilities of each of the letter pairs of a word in the digram. If two consecutive letter pairs have low probability of occurrence ($p \leq .003$), the algorithm considers the common letter to be in error (e.g., x_3 if x_2x_3 and x_3x_4 are judged to be unlikely). An "error" is detected by a trigram if two or more consecutive letter triples have low probability ($p \leq .002$); in this case several rules are used to locate the supposed error. If three trigrams detect the error, it is presumed to be in the position common to all. If there is no position common to all or if two or four or more letter triples have low probability, the digram method is used to localize the error. Processing 100 words with random substitution errors, the digram method successfully located 32% of the errors; the trigram method was successful with 46%. In some more realistic experiments in which the processor was fed data which exemplified the output of a script recognition system, the respective localization rates were 81.5% and 90%.

Sitar corrected an error in position i by selecting that character y which maximized the trigram expression $P(x_{i-1},y,x_{i+1})$. Application of this rule to the random error model resulted in the correction of approximately 30% of the errors (28% for digrams and 32.6% for trigrams) resulting in gross correction rates of only 9 and 15%. In the experiments in which he used data which modelled the output of the cursive script recognition machine, he incorporated information about the likelihoods of various substitutions; this resulted in correction of approximately 73% of the detected and located errors, yielding over-all gross correction rates of 24.5% and 39.3% for this set of errors. Sitar also presents experimental results concerning: improper correction of errors; introduction of a second error into words because of improper localization of the original error; and rules for detection, localization, and correction of deletion errors. Accounting for the spurious errors as well as the error correction techniques, the resultant character recognition

rate was only improved from 90.5 to 93.7%. A three percent improvement is disappointingly small for so much effort in developing algorithms; it was probably so small because the digrams and trigrams were over the entire language and no attempt was made to restrict the vocabulary to a known dictionary. Harmon (1962) describes the above system for the machine reading of handwriting as requiring careful, legible script. Harmon and Sitar (1965) hold a patent on a pattern recognition method and apparatus with a contextual postprocessor.

Edwards and Chambers (1964) perturbed Alt's (1963) moment recognition system with noise to introduce an error rate. The noise was introduced as a normally distributed random variable with mean equal to the means published by Alt and with dispersion as a function of the noise level; they experimented with a number of different noise levels. They then incorporated a subsystem which used digram probabilities to make decisions from among the two characters selected as being the most likely alternatives by the classifier. The result was that, for a noise level of .9, a 10.5% error rate was reduced to 9.4%. They note further that for a noise level of less than .4, the digraph probabilities introduce more errors than they correct. This result is counter to the suppositions of Chu (1971) and Abend (1968) that context could do less for a poor recognition system than it could do for a good recognition system; this was probably because of the weakness of the recognition system design and the digram algorithms.

Carlson (1966) tested n-gram procedures on a data base of over 73000 christening records. He states that, with a scanner error rate of 5%, the trigram replacement technique can correct 95% of the errors, leaving an error rate of 0.25%. He suggests that, since the largest number of trigram errors occur at the beginning and end of words, quadgrams might be useful for these particular cases.

Cornew (1968) designed an algorithm which uses probability digrams to pick the position of a word which is most likely to contain an error, performs substitutions with the most likely alternatives, and makes dictionary lookups. On the average, seven dictionary lookups are required before an error is resolved, whether correctly or incorrectly. With a dictionary of the 1000 most common words of English, his program correctly recovered 73.5% of the randomly selected, randomly placed errors; it also makes 24.1% incorrect recoveries. The remaining 2% had been garbled into valid dictionary words.

McElwain and Evans (1962) describe the Degarbler, a program for correcting machine-read Morse code. The Degarbler scans the output of MAUDE (Gold, 1959) attempting to parse the strings into words. A number of heuristic search procedures are used to find word boundaries. If the Degarbler gets hung up trying to find word boundaries, it stops and goes forward to find the beginning of another word. On a second pass, it rescans the nonsense strings which it gave up on. It uses all of the available letter pair and triple information as keys to those words in the dictionary which contain the pairs and triples, looking for multiple

incidences of references to the same word. It is in the second pass over the strings that the Degarbler corrects the errors made by MAUDE.

Vossler and Branston (1964) experimented with three systems. Using a simulated character recognizer which garbled 20% of its letters, they compared dictionary and digram correction algorithms on a children's primer. The dictionary method reduced the number of errors by 89% and the digram method by 35%. Using the same simulated recognizer with more difficult text, some of whose words were not in the dictionary, they combined the methods and applied digrams to words not found in the dictionary in order to reduce the number of garbles in these words. Of the word occurrences in the text, 25% were not in the dictionary of 3737 word forms. The latter system reduced the net number of errors by 45% although many of the errors in the final text were errors which had been introduced into words which were not in the dictionary.

Thomas and Kassler (1967) combined quadgram occurrence lists with context-dependent syllable grammars. They define an n-gram occurrence grammar as a grammer in which the production rules produce only those strings in which every constituent n-gram occurs in a particular list. They also define a syllable grammar in which a word is defined as concatenations of initial, medial, and final strings (listed in their table VII). They propose the use of the intersection of the two grammars for text processing. No quantitative evaluation is presented.

Riseman and Ehrich (1970) reasoned that, since 42% of all digram occurrences in the English language are zero (Sitar, 1961), they could quantize the probabilities to zero or one and substitute algorithmic methods and greater quantities of digram information for the single huge digram or trigram which had been used previously. The storage is saved because instead of a floating point word's being required to store a probability, a single bit is sufficient. Thus, it became economical to incorporate information about many positional dependencies. For example, although it is known that adjacencies provide the greatest quantity of contextual information, non-contiguous positions do provide some useful information. Then, through the use of appropriate algorithms errors can be located and corrected (Riseman and Ehrich, 1971; Ehrich and Riseman, 1971; Riseman and Hanson, 1974; Ehrich and Koehler, 1975; and Hanson, Riseman, and Fisher, 1975). This particular technique is explained in greater detail and generalized in chapter III.

C H A P T E R I I I

BACKGROUND: THE METHOD OF BINARY n-GRAMS

3.1. INTRODUCTION

In this chapter the method of binary n-grams is reviewed. First, the methods and techniques of applying binary n-grams to text are presented. Then, the achievements and results of a number of research papers which employed binary n-grams are reviewed.

Riseman and Hanson (1974) applied binary n-grams in a contextual post-processor which processed simulated classifier output. A subsequent paper applied additional postprocessing to resolve ambiguities which had been rejected by this postprocessor (Hanson, Riseman, and Fisher, 1975). Riseman and Ehrich (1971) used binary n-grams in a contextual postprocessor (CPP) of substitution set data. The substitution set techniques were later elaborated and applied to cursive script recognition (Ehrich, 1973; and Ehrich and Koehler, 1975). Finally, an algorithm for selecting a good subset of binary digrams was explored by Ehrich and Riseman (1971).

3.2. BACKGROUND

Strings of data are input to the contextual post-processor by a character recognizer. The task of the CPP is to determine whether or not each input string is a word. If it is determined that a particular string is not a word, the CPP must attempt to determine what is wrong with the string and then, if the problem is among some predetermined set of problems, the CPP must try to correct it.

It is a relatively easy matter to determine whether or not a string of characters is a word; the CPP only has to look up the string in the dictionary. If the string is not in the dictionary, an error has been detected. Suppose the CPP detects an error in an m letter word. How is the error to be corrected? The obvious, straightforward, scheme is to replace each character of the word by some other character: twenty-five alternates for the first position, twenty-five for the second, etc. If the string represents an m letter dictionary word with one substitution error, this method is guaranteed to resolve the error after 25m dictionary lookups. If the word has two substitution errors there are 25 $\binom{m}{2}$ substitutions and dictionary lookups to be made. Some better means must be found for locating and correcting errors. In addition to the computational difficulty of searching the dictionary it might be impossible to store so large a dictionary.

3.3. THE METHOD OF BINARY n-GRAMS

3.3.1. Binary Digrams

A positional binary digram is a matrix D_{ij} whose elements $d_{ij}(a,b)$ are 1 if and only if the letters C_a and C_b occur in positions i and j, respectively, of some word in the dictionary; 0 otherwise. Thus, a binary digram possesses some information about the words in the dictionary. Without loss of generality we can require that $i < j$ since D_{ji}

would only be the transpose of D_{ij} anyway. Similarly, a positional binary trigram is a binary array T_{ijk} whose elements $t_{ijk}(a,b,c)$ are 1 if and only if the characters C_a, C_b, and C_c occur in positions i, j, and k, respectively, in some word in the dictionary. For ease of notation, trigrams will frequently be referred to with the initial D in the text; when it is necessary to distinguish among the various types of n-grams, the context will be sufficient.

Binary n-grams differ from probability n-grams in a number of ways. The techniques which employ probability n-grams (reviewed in the last chapter) generally consider only adjacent positions as indices of n-grams. The methods of binary n-grams do not ignore the far-reaching relationships of letters several positions apart. Probability n-grams were usually used without regard to position; for example, the same n-gram which was used for positions 1, 2, and 3 would be used for positions 2, 3, and 4, and 4, 5, and 6, etc. These n-grams are defined and discussed later in this section.

Binary digrams require 26^n bits of storage while probability digrams require 26^n floating point words; the storage saved per n-gram permits a system to use a far greater number of binary n-grams than probability n-grams and, in this manner, bring a far greater quantity of contextual information to bear upon the problem. (Toussaint (1974) claims that the floating point words used in Markov methods require only four bits; however, he provides no quantitative comparison of methods.) Also, operations on D_{ij} are boolean in nature while the probability digram P_{ij} requires floating point arithmetic.

Since binary n-grams are quantized versions of probability n-grams, they lack information about relative likelihoods; for example, $d_{12}(e,n) = d_{12}(e,q)$ but $p_{12}(e,n) > p_{12}(e,q)$. The comparative information could be used for determining the letter most likely to be in error (Sitar, 1961). Thus the binary digram provides some convenient computational considerations in exchange for a loss of comparative information. There are other differences which arise because of the boolean nature of the arrays. Probability n-grams are usually defined for the entire language: Sitar used n-grams which had been compiled for the English language; Raviv gathered language statistics from his training data but allowed a finite, non-zero probability for those combinations of letters which had not occurred in the training set. Obviously, n-grams would lose all of their value if they allowed a non-zero likelihood for those combinations which never occurred. (The algorithms could allow such a finite, non-zero probability but the binary n-grams themselves would provide no information if they did because every quantized probability would be 1.)

Binary n-grams differ in the sense that they refer to a definite portion of the language. The portion of the language is a fixed, predefined dictionary and, therefore, a fixed, well-defined application domain. Because of the well-defined domain, the zeroes of an n-gram indicate that the corresponding n-tuples of letters never occurred in the relevant portion of the language. Erroneous positions are located

with some sense of absoluteness; when a word is determined to have an error, the candidate positions are selected with the same sort of boolean operations used for detecting errors. It must now be shown that the loss of information is not a handicap since the small size of the digrams allows a system to incorporate more D_{ij} matrices than it would if it were using probability digrams.

3.3.2. Methods of Detection, Location, and Correction Of Substitution Errors

The applicabion of binary n-grams to text processing was initially explored by Riseman and Hanson (1974). They compared contextual post-processors which used several different n-gram techniques for detection and correction of substitution errors.

Let us denote the application of an n-gram D to a sequence of characters X as D(X) where D(X) is defined as $d_{ij}(x_i,x_j)$ and X is a string $x_1x_2\ldots x_m$ such that each x_i is a character of the alphabet. Given a set of n-grams $\mathcal{D}$, a string X is an admissible sequence if for each $D \in \mathcal{D}$, $D(X)=1$. If $X=x_1x_2\ldots x_m$ has a single substitution error in position r then, for some character y, there is a dictionary word $Y=x_1x_2\ldots x_{r-1}\ y\ x_{r+1}\ldots x_m$ which is the correct spelling of X. For example, "HOVSE" is a single substitution error version of "HOUSE" with r=3.

3.3.2.1. Single Error Detection

Given the set of positional n-grams $\mathcal{D}$, an error has occurred if the "dictionary syntax," as represented by the n-grams of $\mathcal{D}$, has been violated. Given a sample word X, $X=x_1x_2\ldots x_m$, and each x_i is a member of the alphabet, then X is not admissible if

$$\prod_{D\in\mathcal{D}} D(X) = 0 \; . \qquad (3.1)$$

That is, since the product is zero if at least one D(X) is zero, some digram in the set $\mathcal{D}$ has rejected X as inadmissible.

However, some words in error will not be detected by this test. As a simple example, consider the four word dictionary "SOLE," "CARE," "CULL," and "CORN." The word "COLE" has a digram undetectable error because each of its letter pairs occurs elsewhere in the dictionary. (The digrams "-OL-," "-O-E," and "--LE" occur in "SOLE," and the digrams "C--E," "C-L-," and "CO--" occur in the words "CARE," "CULL," and "CORN," respectively. However, "COLE" is not a trigram undetectable error because the trigrams "C-LE," "CO-E," and "COL-" do not occur anywhere else in the dictionary.

3.3.2.2. Location of Single Errors

Define the function Pos such that $\mathrm{Pos}(D_{ij}) = \{i,j\}$; that is, when Pos is applied to a particular n-gram, it yields the set of positional indices of that digram.

If there is a single substitution error in position r of X and the product in (3.1) is zero then

$$r \in \bigcap \{\mathrm{Pos}(D) \mid D \in \mathcal{D} , D(X)=0\} \; . \qquad (3.2)$$

If the product (3.1) is zero, some set of digrams has rejected the word X. The incorrect position r must be one of the positions of each rejecting digram. Else, if $D_{ij}(X)=0$ and $r\neq i$ and $r\neq j$, something is wrong with the characters in positions i and j and the hypothesis of exactly one error, a substitution in position r, must be false. The above rule also applies to binary trigrams.

There are essentially three different situations which may occur when using n-grams to process a word with a single substitution error. The cases are best illustrated by using trigrams. In the first case the position of the error is easily pinpointed. As a simple example, suppose the rejecting trigrams are T_{124} and T_{156}; it is clearly indicated that if X has a single substitution error, it is in position one. Correction can then be attempted as discussed previously. The second case occurs when the position is not clearly indicated. If only T_{145} rejects X, under the hypothesis of a single substitution error, one only knows that the error is in either of positions one, four, or five. If T_{145}, T_{124}, and T_{146} reject the word, then it is known that a single error must be in either of positions one and four. Finally, if the input word in error is an admissible sequence, it contains an undetectable error; note that the statement (3.2) is still true.

3.3.2.3. Error Correction

Binary digrams can be used to attempt correction of the errors which are detectable. Consider a six-letter word, $x_1x_2x_3x_4x_5x_6$, with x_3

the only character that is incorrect. If the binary digrams used are sufficiently sparse[1], several of them may detect the error, D_{13}, D_{23}, D_{34}, D_{35}, and D_{36}. If at least two of them detect an error (say D_{23} and D_{35}), the position is fixed by noting that position 3 appears as one of the indices of each of the digrams. If some character is to replace x_3, then each of the five digrams contains some information about the identity of the replacement. The row $d_{13}(x_1,*)$ contains a 1 in all locations in which the corresponding character is allowable in position 3, given x_1 in position 1; similarly, the column $d_{34}(*,x_4)$ contains a 1 in all locations in which the corresponding character is allowable in position 3 given that x_4 is in position 4. If the proper rows and columns of the five digrams are logically intersected, all available information will have been utilized in error resolution. If the j-th entry in

$$d_{13}(x_1,*) \wedge d_{23}(x_2,*) \wedge d_{34}(*,x_4) \wedge d_{35}(*,x_5) \wedge d_{36}(*,x_6)$$

is 1, then the substitution of C_j for x_3 will be acceptable to all digrams and produce an admissible sequence according to the dictionary syntax.

If there is more than one entry in the product vector which is a 1, there is an ambiguity and the word is either rejected or subjected to further processing, such as a dictionary lookup which may or may not

[1] Of course, these algorithms are heavily dependent upon the sparseness of the n-gram arrays. Sitar (1961) reported that only 50% of all possible digrams and 13.2% of all possible trigrams occurred in a sample of English text. In the n-grams used in chapter V, we found that 12 to 30% of the positional trigrams and 6% of the non-positional quadgrams occurred in our sample of 19196 city names.

resolve all of the ambiguities. As an example of such an ambiguity, suppose that in correcting "HOVSE" it is discovered that both U and R fit the third position; the correction cannot be resolved and the word must be rejected.

3.3.2.4. Some Examples

Figures 3.1.a-f present a fixed dictionary and several examples of words which are assumed to contain single substitution errors. Each of the sample words contains an error which is different with respect to the contextual postprocessing algorithms; they are: in figure 3.1.b, an undetectable error; in 3.1.c, an unambiguously correctable error; 3.1.d, an ambiguous correction which is unambiguously located; 3.1.e, an ambiguous correction whose position is also ambiguous; and, in 3.1.f, an unambiguous correction whose position was initially ambiguous. For the sake of the illustration we are using all possible binary digrams; generally, however, it has been found that trigrams provide the CPP with better correlative power.

The undetectable error ("FELD") in figure 3.1.b is digram undetectable (i.e., undetectable by binary digrams) because each of its digrams (letter pairs) occurs in at least one word in the dictionary.

Figure 3.1.c presents an example of an unambiguously correctable error. To clarify the correction method, the process of intersecting the appropriate rows and columns of the binary digrams is illustrated in

CELT
FELT
FOLD
MALT
MELT
MOIL
SLIT

Figure 3.1.a The dictionary used in Figures 3.1.b-f

Input word: FELD

$d_{12}(F,E) = 1$	$d_{23}(E,L) = 1$
$d_{13}(F,L) = 1$	$d_{24}(E,D) = 1$
$d_{14}(F,D) = 1$	$d_{34}(L,D) = 1$

$$\prod_{D \in \mathcal{D}} D(FELD) = 1$$

Figure 3.1.b An example of an undetectable error, given the dictionary of Figure 3.1.a and the CPP which uses all positional binary diagrams.

Input word: FEIT

$d_{12}(F,E) = 1$	$d_{14}(F,T) = 1$	$d_{24}(E,T) = 1$
$d_{13}(F,I) = 0$	$d_{23}(E,T) = 0$	$d_{34}(I,T) = 1$

The error is in position 3.

$d_{13}(F,*) \wedge d_{23}(E,*) \wedge d_{34}(*,T)$

$$\begin{bmatrix}0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\1\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\end{bmatrix} \wedge \begin{bmatrix}0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\1\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\end{bmatrix} \wedge \begin{bmatrix}0\\0\\0\\0\\0\\0\\0\\0\\1\\0\\0\\1\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\end{bmatrix} = \begin{bmatrix}0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\1\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\end{bmatrix} \Rightarrow \{L\} \Rightarrow \text{FELT}$$

Figure 3.1.c Example of an unambiguously correctable word.

Input word: SELT

$d_{12}(S,E) = 0$ $d_{14}(S,T) = 1$ $d_{24}(E,T) = 1$

$d_{13}(S,L) = 0$ $d_{23}(E,L) = 1$ $d_{34}(L,T) = 1$

The error is in position 1.

$d_{13}(*,E)$ $d_{12}(*,L)$ $d_{14}(*,T)$

$$\begin{bmatrix}0\\0\\1\\0\\0\\1\\0\\0\\0\\0\\0\\0\\1\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\end{bmatrix} \wedge \begin{bmatrix}0\\0\\1\\0\\0\\1\\0\\0\\0\\0\\0\\0\\1\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\end{bmatrix} \wedge \begin{bmatrix}0\\0\\1\\0\\0\\1\\0\\0\\0\\0\\0\\0\\1\\0\\0\\0\\0\\0\\1\\0\\0\\0\\0\\0\\0\\0\end{bmatrix} = \begin{bmatrix}0\\0\\1\\0\\0\\1\\0\\0\\0\\0\\0\\0\\1\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\\0\end{bmatrix} \begin{array}{l} \Rightarrow \{C,F,M\} \\ \Rightarrow \text{CELT, FELT, or MELT} \end{array}$$

Figure 3.1.d An example of an ambiguous correction.

Input word: CALT

$d_{12}(C,A) = 0$ $d_{23}(A,L) = 1$

$d_{13}(C,L) = 1$ $d_{24}(A,T) = 1$

$d_{14}(C,T) = 1$ $d_{34}(L,T) = 1$

If there is a single substitution error in CALT, it must be in either position 1 or position 2.

Attempting to correct position 1:

$d_{12}(*,A) \wedge d_{13}(*,L) \wedge d_{14}(*,T) \Rightarrow \{M\}$

Attempting to correct position 2:

$d_{12}(C,*) \wedge d_{23}(*,L) \wedge d_{34}(*,T) \Rightarrow \{E\}$

The correct form is either MALT or CELT; therefore the word is rejected.

Figure 3.1.e An example of an ambiguous position which results in an ambiguous correction and, therefore, a rejection.

Input word: MLIT

$d_{12}(M,L) = 0$ $d_{23}(L,I) = 1$

$d_{13}(M,I) = 1$ $d_{24}(L,T) = 1$

$d_{14}(M,T) = 1$ $d_{34}(I,T) = 1$

If there is a single substitution error in MLIT, it is in either position 1 or position 2.

Attempt correction of position 1:

$d_{12}(*,L) \wedge d_{13}(*,I) \wedge d_{14}(*,T) \Rightarrow \{S\}$

Attempting to correct position 2:

$d_{12}(M,*) \wedge d_{23}(*,I) \wedge d_{24}(*,T) \Rightarrow \{\ \}$

i.e., there can be no error in position 2 because no letter fits there, given the context of the other characters.

Figure 3.1.f An example of an ambiguous position which is resolved and corrected.

this and the next examples. Note that the third vector, which represents $d_{34}(*,T)$, contains more than one 1; the extra 1 is "anded out" by the other vectors. In actual practice each vector often has several 1's, most of which are eliminated by intersection with the other vectors. Figure 3.1.d presents an example in which the position is determined but the error is not corrected.

Figures 3.1.e and 3.1.f deal with the case of the ambiguous position; that is, the position of the error is not clearly indicated. In the example of figure 3.1.e, no correction is made because there is a possible correction for each of the positions. The example of figure 3.1.f is corrected because there is no character which can be placed in position 2 to correct the word; therefore, the error must be in position one and correction is made. Of course, it is also possible for the position to be resolved in this manner without resulting in a correction.

3.3.3. Locating a Double Substitution Error

If X is a word with substitution errors in positions q and r and is a set of n-grams, then

$$\text{for each } D \in \mathcal{E} \text{ such that } D(X)=0 \qquad (3.3)$$
$$q \in \text{Pos}(D) \text{ or } r \in \text{Pos}(D).$$

That is, if an n-gram rejects X and if X is a word with two substitution errors, then it must be that $q \in \text{Pos}(D)$ or $r \in \text{Pos}(D)$ since otherwise there would be no reason for D to have rejected X other than the possibility that the hypothesis of exactly two errors--substitution errors in

positions q and r--is wrong. For example, let X="NOVSE," an incorrect spelling of the dictionary word "HOUSE"; q=1 and r=3. It is not possible that $D_{245}(X)=0$ since there is nothing wrong with any of the positions 2, 4, and 5 and neither $1 \in \{2,4,5\}$ nor $3 \in \{2,4,5\}$.

In general, if X has s substitution errors, in positions $m_1, m_2, \ldots, m_s$, then

$$\text{for each } D \in \mathcal{E} \text{ such that } D(X)=0$$
$$\text{there exists } i,\ 1 \leq i \leq s,\ m_i \in \text{Pos}(D).$$

The reasoning here is similar to the above since under the hypothesis of s substitution errors, there cannot be a rejecting n-gram D which does not use at least one of the m_i. Note that it has never been said that each of the m_i must belong to any of the rejecting n-grams.

If it is found that a word has more than one substitution error, an attempt is made to correct it under they hypothesis of two substitution errors. The same approach to correction is taken as in the single error problem; correction is first attempted in each pair of positions which might contain the errors.

One way to determining the positions which might contain substitution errors is illustrated by Tables 3.1 and 3.2. In Table 3.1, D_{123}, D_{134}, D_{135}, and D_{145} have all detected errors. Each row of Table 3.1 marks the positions of one of the trigrams that detects an error. It should be clear that the set of positions in error must account for at least one * in each row; otherwise the trigram in that row could not

TABLE 3.1

EXAMPLE OF FIXING THE POSITION OF ERRORS VIA POSITIONAL BINARY TRIGRAMS

	1	2	3	4	5	6
D_{123}	*	*	*			
D_{134}	*		*	*		
D_{135}	*		*		*	
D_{145}	*			*	*	

Possibilities for one error-- {1} .
Possibilities for two errors-- {1,2} , {1,3}, {1,4}, {1,5}, {1,6}, {3,4} , {3,5} .

TABLE 3.2

EXAMPLE OF FIXING THE POSITION OF ERRORS VIA POSITIONAL BINARY TRIGRAMS

	1	2	3	4	5	6
D_{124}	*	*		*		
D_{135}	*		*		*	
D_{156}	*				*	*
D_{236}		*	*			*
D_{345}			*	*	*	
D_{456}				*	*	*

Possibilities for two errors-- {2,5} .

have detected an error. Thus, if a single error occurred, it must be in position one. However, if two errors occurred, there are several pairs of positions which could account for the particular set of rejecting trigrams. This can quickly be determined by enumeriating all of the $\binom{m}{2}$ pairs of positions to determine which could contain two errors. In Table 3.2, six positional trigrams have detected errors and it is not possible that the word could be a one error word; there is only one pair of positions which could contain a double substitution error. Of course, there could be more than two errors. This algorithm is similar to some of the algorithms for selection of essential prime implicants in switching theory and to various set covering algorithms in graph theory (McCluskey, 1965; Even, 1973).

3.3.4. Correcting Multiple Substitution Errors

The condition which must exist in order for there to be substitution errors in positions m_1, m_2, ..., were presented in (3.3) and (3.4). Let us now consider their implementation. The error detection system provides the indices of the trigrams which reject the word X (i.e., the trigrams for which T(X)=0; this information can then be used to hypothesize the indices of the positions which might contain errors.

The first assumption of the CPP is that the word input to it has no errors. Once this hypothesis is contradicted, the CPP attempts another hypothesis; is it possible that the word has a single substitution error? If there are no admissible one error substitutions, the CPP assumes that the word has two errors and attempts correction.

The method of single error correction using binary digrams--row intersection of the proper rows (or transposed columns)--is also used with trigrams. The process used for correction of two error words will now be outlined. Suppose that positions two and five are both suspected of containing substitution errors. Each trigram T such that $2 \in \mathrm{Pos}(T)$ but $5 \notin \mathrm{Pos}(T)$ is applied to find a set of characters which might fit position 2 given the context of the rest of the word other than position 5. The same procedure is then applied to find a set of letters which fit position 5. The set of trigrams such that $\{2,5\} \in \mathrm{Pos}(T)$ are then applied in an attempt to find the set of characters which fit both positions given the context of the rest of the word. If any of the three sets sought in this procedure is empty, positions 2 and 5 cannot both contain substitution errors. If there is more than one candidate set of corrections available, the word is then rejected.

3.3.5. Transposition Errors

In the process of locating, detecting, and correcting double substitution errors, transposition errors can also be located and corrected. In the simple case, if only one pair of positions could be in error, if they are adjacent, and if there is only one pair of letters which could fit the word, it will be corrected whether it be a transposition error or a double substitution error. If there are several candidate pairs of

letters and only one in adjacent positions, and if there is a correction for a transposition available, the possibility of a transposition error will have to be weighted against the possibilities of a double substitution. If a transposition is more likely than a double substitution error, it will always be corrected if it is detected.

3.3.6. Non-Positional n-Grams

A non-positional binary digram is a binary array C whose elements c(a,b) are one if and only if the characters C_a, and C_b occur in adjacent positions in some word in the dictionary. Here, the information from the different adjacent triples of positions is combined under union. A non-positional n-gram carries information about more of the word but it is less specific about any particular position in the word. Usually these n-grams require 27^n bits of storage because they are also used to check for end of word conditions (the blank is the 27-th character). The definition and methods extend easily to binary trigrams and quadgrams. Riseman and Hanson (1974) compared the results of using non-positional digrams, trigrams, and quadgrams in their experiments with n-grams.

In the next chapter, we shall need a consistent definition of the position function, Pos, so we shall now present a definition of Pos as it applies to non-positional n-grams. Let C be a non-positional digram. It is meaningless to speak of Pos(C) since C is non-positional. However, if C is applied to X and C(X)=0, a method is needed for referring to the positions indicated. Let $C'_i=C$ be a unique name for C which is the name used when C is to be applied to positions i and i+1 of X. Then, define $Pos(C'_i)$ as $\{i,i+1\}$. It then follows that all of the statements concerning detection, location, and correction of errors are consistent with this definition. The definition of Pos extends naturally to use with non-positional n-grams in general.

3.4. BINARY n-GRAMS IN A CPP

Riseman and Hanson (1974) compared several CPP designs which used binary n-grams. Using dictionaries of 300, 800, 1300, and 2755 six-letter words, they compared the results of implementing a CPP with 15 positional binary digrams, 20 trigrams, one non-positional digram, a non-positional trigram, a non-positional quadgram, a few subsets of the set of 20 positional binary trigrams, and a dictionary look-up algorithm.

3.4.1. Comparing n-Gram Methods

The experiments compared and contrasted these methods using the detection, correction, and error rates for single, double, and triple error words. The methods were ranked consistently in nearly every one of the comparisons. For example, the error detection and correction rates for one and two error words when the dictionary had 2755 words are presented in Table 3.3.

TABLE 3.3

A COMPARISON OF n-GRAM TECHNIQUES

n-Gram Technique	Bits of Storage	One Substitution		Two Substitutions	
		Detected	Conditional Corrections	Detected	Conditional Corrections
Dictionary	82650+	99.7%	84.1	100.%	46.8
Non-Positional Quadgram	531444	96.9	47.8	96.9	7.5
20 Positional Trigrams	351520	98.6	62.4	99.8	34.1
6 Positional Trigrams	105456	93.8	26.8	98.8	4.6
4 Positional Trigrams	70304	88.2	19.4	98.2	2.4
Non-Positional Trigram	19683	78.0	2.8	91.6	0
15 Positional Digrams	10140	69.6	.5	87.6	---
Non-Positional Digram	729	32.5	0.0	49.4	---

Notes:

The dictionary had 2755 six-letter words.
---is used to indicate that correction was not attempted.

A weak rule of thumb is that the power of the CPP is related to the quantity of storage required for the binary arrays. However, the exceptions to the rule are at the top end of the rankings. Table 3.3 also presents the number of bits required by each technique. It should be noted that the rankings are in the context of dictionaries of n-letter words; Ehrich and Riseman (1971) compared one non-positional digram to 21 positional digrams for seven-letter words and found the digrams to be superior.

Using a dictionary of 2755 six-letter words and 20 positional trigrams, the CPP was able to detect that errors existed in 98.6% of one error words and 99.8% and 99.7% of two and three error words, respectively. They were subsequently able to correctly modify 62.4% and 34.1% of the one and two error words in which errors had been detected.

They also observed that 76.4% of the rejected one error words had only two or three ambiguous single error forms; from this they surmised that additional processing to resolve ambiguities would yield higher correction rates at the expense of some small additional error rate. Thus they suggest that, if the classifier had a character error rate of 10% (leading to a word error rate of 46.86%), their CPP would properly correct 52.8%, reject 44.6%, and err on 2.4% of the errors. This would yield a system (classifier plus CPP) word recognition rate of 75.2%. To resolve ambiguities, they assume that guesses could be forced correctly 80% of the time for those ambiguous forms which required only two choices and at a rate of $(.80)^{n-1}$ for other combinations involving n choices. Then, with the above classifier error rate of 10%, they conjectured that the system could correctly recognize 88% of the words it would process.

3.4.2. Additional Postprocessing

Hanson, Riseman, and Fisher (1975) researched the problem of resolution of ambiguities (rejections) using various other information available to the postprocessor. The research goal in this paper was to resolve the rejections due to ambiguities. A Bayes classifier was built (Fisher, Hanson, and Riseman, 1974) to classify handprinted characters (Munson, 1968). Words which simulated text were then randomly generated using a set of test patterns from Munson's data. They were processed by the classifier and formed the input to the CPP. A module was attached to the CPP. The goal of the module was to correctly resolve those ambiguities which could not be resolved by the CPP. The input to the module was the questionnable form of the word as output by the classifier and the set of ambiguous forms--for any word hypothesized to have three or fewer substitution errors.

The various resolution schemes used were: (1) use of substitution probabilities which had been acquired on the training set; (2) use of the original Bayes classifier probabilities associated with the patterns thought to be in error; (3) use of specialist features selected for their ability to resolve the ambiguities in question; and (4) combinations of each.

The research also studied the results of integrating the classifier, CPP, and the decision module. For a six-letter word error rate of 45.8% (a character error rate of 9.7%), the system achieved a word error rate of 2.7% with less than 0.1% rejections. With some selectiveness in the decision process--i.e., requiring confidence in decisions--a word error rate of 1.7% and a reject rate of 2.7% were achieved. Incorporating a dictionary as the ultimate arbiter to reduce the number of errors and ambiguities, the rates just cited were reduced to 1.2% error and 0.1% rejections for the case of forced decisions and 0.9% error and 0.7% error for the case in which some confidence was desired.

In the system simulated there was a total of 4528 words rejected by the CPP to the decision module; the methods were compared on the basis of their ability to resolve these ambiguities. The general result was that the Bayes probabilities were the most successful single method; the specialist features were a close second; and the substitution probabilities, third. When the methods were forced to make a decision they correctly recognized 91.0, 87.3, and 76.8%, respectively, of the 4528 rejected words. Combining the methods provided marginally better results. The best was the combination of the Bayes classifier decision likelihoods and the substitution set probabilities; but this method only achieved a recognition rate of 91.9%.

The final system recognition rates, 97.2% of the words processed and 98.5% when using a dictionary, are very high for a system which uses so large a dictionary and which has so large an initial error rate. A pattern recognition system which is to use such a decision module with a CPP would have a very simple addition to its design; it would only have to transmit from the classifier to the CPP the classifier likelihoods

which are assoicated with its decisions.

3.5. BINARY n-GRAMS APPLIED TO SUBSTITUTION SETS

Riseman and Ehrich (1971) experimented with a different processor design. Their modified character recognizer outputs sets of characters (substitution sets) instead of single character decisions. For example, if it were to output a substitution set of four characters for the input character "X," the set might be $\{K,V,X,Y\}$ (or even $\{A,K,V,Y\}$ if the recognizer made errors.)

3.5.1. Postprocessor Design

Their contextual postprocessor used the binary digrams as a set of snytax rules to determine which sequences of combinations from the subsitution set were admissible sequences. An admissible sequence is a sequence of characters $x_1,x_2,\ldots,x_m$, which conforms to all of the rules of the dictionary syntax as represented by the binary digrams. The CPP of Riseman and Ehrich applies its digram set to the string of substitution sets to determine the set of all admissible sequences. The admissible sequences are then looked up in the dictionary. Each admissible sequence which is found in the dictionary is an admissible d-sequence. (The words in the dictionary are called d-sequences.)

A simple example of the use of binary digrams in this experimental design is given in Figure 3.2, where the alphabet consists of four letters, the dictionary has 6 three-letter words, the substitution sets have two letters, and the dictionary syntax uses 3 digrams. Of the 8 three-letter sequences which can be formed, only two are admissible and only one is in the dictionary.

DICTIONARY

$$\begin{bmatrix}\text{DAB}\\\text{BAD}\\\text{CAD}\\\text{DAD}\\\text{CAB}\\\text{ADD}\end{bmatrix}$$

DICTIONARY SYNTAX

$$\begin{bmatrix}0&0&0&1\\1&0&0&0\\1&0&0&0\\1&0&0&0\end{bmatrix}\quad\begin{bmatrix}0&1&0&1\\0&0&0&0\\0&0&0&0\\0&0&0&1\end{bmatrix}\quad\begin{bmatrix}0&0&0&1\\0&0&0&1\\0&1&0&1\\0&1&0&1\end{bmatrix}$$

SUBSTITUTION SETS

$$\begin{bmatrix}\text{B}\\\text{A}\end{bmatrix}\quad\begin{bmatrix}\text{A}\\\text{C}\end{bmatrix}\quad\begin{bmatrix}\text{D}\\\text{B}\end{bmatrix}$$

ADMISSIBLE SEQUENCES

$$\begin{bmatrix}\text{BAD}\\\text{AAD}\end{bmatrix}$$

ADMISSIBLE D-SEQUENCES

BAD

Figure 3.2 Example of the use of binary digrams with substitution sets

Note that while there are 8 three-letter sequences which can be formed, it is not necessary for a sequential processor to completely form more than four of them to check for admissibility since D_{12} can be applied to the partial sequences "BD-" and "AC-" to inhibit further development of these sequences; thus, four of the eight sequences do not have to be completely checked out.

If more than one of the admissible sequences are admissible

d-sequences, the word must be either: (1) rejected, (2) subjected to further processing, or (3) output for human intervention with some auxiliary data such as the set of admissible d-sequences and the surrounding resolved text.

Riseman and Ehrich conclude their theoretical development by presenting a few formulas for the expected number of admissible d-sequences given the size of the substitution sets, the particular set of words in the dictionary and their respective a priori probabilities.

3.5.2. Experimental Results

They then present a number of experiments which demonstrate the effectiveness of the method. They experiment with various seven-letter word dictionaries of from 50 to 350 words. With five letters in the substitution sets and a 350 word dictionary, the postprocessor was able to correctly resolve 97.02% of the input strings--assuming that the correct letter was always in each substitution set and that the rest of the members of the substitution sets occurred randomly. They also experimented by varying the size of the substitution sets and with differing strategies which required forced decisions in some cases when many admissible d-sequences were encountered.

3.5.3. Recognition of Cursive Script

Ehrich (1973) applied binary digrams and substitution sets to the problem of dynamically recognizing cursive script. Because the recognition of handwriting presents a segmentation problem, Ehrich marked each initial downstroke of each character by a segment mark and each major downstroke with a presegment mark. The task of his CPP was to determine which presegment marks were segment marks. To accomplish this, a word with four downstrokes required four substitution sets whether the word had two, three, or four letters, one substitution set per presegment mark. Any character is automatically associated with the number of presegment marks which it requires. Thus, if an "a" (two presegment marks) is in the first substitution set, its selection for the search tree would force it to be associated with a ∅ (indicating no presegment marks) from the second substitution set. (A few characters are associated with different numbers of presegment marks.) Other characters which require only one presegment mark would always force the search routine to ignore any ∅ in the second substitution set. Thus, the search routine always associates each character with the proper number of substitution sets. To avoid saturation of the digrams used, Ehrich proposes to use a different set of digrams for each set of words of different lengths in the dictionary.

Ehrich and Koehler (1975) describe the cursive script recognition system which implements the above scheme in an on-line pen tracking system. Using a dictionary of 300 seven-letter words and testing on the training data, they achieved a correct recognition rate of 98.7% with 1.3% rejection and .0004% error; without any contextual statistics the recognition rate was only 16.5% on the same data when the recognizer was forced to output its best guesses. When the contextual system was applied to the handwriting of a stranger--a person on whose handwriting

it has not been trained--it correctly processed 70.4% of his words, rejected 29.4% and erred on .17%.

3.6. A DIGRAM SELECTION ALGORITHM

Ehrich and Riseman (1971) investigated the problem of selecting a good subset of a set of digrams. They present their algorithms and demonstrate the algorithms's effectiveness with some experiments. The set of 21 digrams for seven letter words is a highly redundant set. The information present in D_{13} overlaps with the information in D_{12} and D_{23}. If the boolean sum $\sum_{i=a} d_{12}(a,i)d_{23}(i,b)$ is zero, then $d_{13}(a,b)=0$. Therefore, given that D_{12} and D_{23} have already been selected, the additional information contained in D_{13} is provided by those elements of D_{13} which are zero when $D_{12}D_{23}$ is not; i.e., all the ones of $D_{12}D_{23}-D_{13}$. ($D_{12}D_{23}$ is the product of the two boolean matrices D_{12} and D_{23} and is formed by using boolean addition and boolean multiplication of the elements of the matrices.) Ehrich and Riseman exploit this fact through its many complex combinatoric forms--many of the interdependencies involve more than just a few digrams. Their experimental results demonstrate that, using 11 of the 21 possible digrams for seven-letter words, their algorithm compares favorably against both a randomly selected set of 11 and a set of 11 selected by a simpler heuristic.

3.7. DISCUSSION

An important consideration with these techniques is the quantity of storage required; the twenty positional trigrams used by Riseman and

Hanson (1974) and Hanson, Riseman and Fisher (1975) used 351,521 bits of storage (5-25k depending on the machine being used). In view of the continuing trend toward cheaper memories in the industry, this objection becomes minor.

It is apparent that there are several misconceptions which might be formed about the techniques. Toussaint (1974) points out a few of these. It is not necessary for a CPP to use all of the n-grams which are possible. It is felt that for longer words, since the redundancy per letter is greater and the dictionaries are relatively smaller, the number of n-grams to be used could be reduced to much less than the combinatoric maximum of $\binom{m}{n}$. It appears that some sort of inverse relationship exists between n-grams required and word length. It is known that even a dictionary is insufficient for processing short words (Szanser, 1971); n-grams must also be shown that word length is related to CPP effectiveness.

Another possible complaint about storage requirements is that it is necessary to use a different set of n-grams for every different word length in the dictionary. This is not true since one could easily conceive of a system with one standard size of word (padded with blanks) for which a single set of n-grams is used; or even sets of n-grams for short words, sets for medium length words, and set for long words. From the works of Riseman and Hanson (1974) and Hanson, Riseman, and Fisher (1975) we see that a dictionary is not a necessary part of the system.

Toussaint (1974) also tried to compare the digram methods to the various Markov methods on the basis of storage alone. Obviously the

positional binary n-gram methods have produced superior results and comparisons will have to be based on the relative merits of problem design and goals rather than storage alone.

Responding to Toussaint's claim that the Markov methods may be adequately applied with only four bits per transition probability, we must point out that his comparison was totally inadequate. He stated that only four bits were necessary for each of the transition probabilities and, therefore, that only $4x26^2$ bits are needed to store the probabilities. He then points out that this is far less than the $\binom{12}{2}$x26 bits required for the use of all possible binary digrams for all words up to 12 letters in length. The published results, cited in the previous chapter, concerning use of trigrams which reduced a character error rate of 2% to .6% are not as remarkable as those of Hanson, et al., (1975) in which a character error rate of 9.7% was reduced to less than .43%. Because of this, we must dismiss the objection; however, we add that, should it be useful to do so, such four-bit-per-element n-grams could be used by our algorithms.

In the forthcoming chapters, new methods of binary n-grams will be presented. The methods will permit efficient solution of the multilength word problem and will introduce algorithms for locating and correcting insertion, deletion, merger, and split errors with n-grams.

CHAPTER IV

NEW DEVELOPMENTS IN THE METHODS OF BINARY n-GRAMS

4.1. INTRODUCTION

In chapter III, the methods of binary n-grams were presented together with techniques for detection, location, and correction of single and multiple substitution errors and transposition errors. In the first part of this chapter the concept of the binary n-gram is extended to include notions of positional anchoring. With this additional concept the characters of a word are viewed in the context of their distance from either or both ends of the word. The positional anchoring allows improved methods of correction of additional types of errors because it provides a new interpretation of the rejection; a rejection $D_{ijk}(X)=0$ implies that "at least one of those characters does not look right in one of those positions." Prior interpretation of this has been used to correct incorrect characters; application of n-gram techniques to other error types allows a system to consider the possibility that a character may be correct but in the wrong position. In this chapter n-grams are applied to the detection, location, and correction of insertion and deletion errors as well as to some similar types of errors, splits and mergers, which are defined in the chapter.

4.2. NEW BINARY n-GRAM DEFINITIONS

In this section the concept of positional anchoring is presented. The systems that were discussed in the last chapter used sets of input

words which were all the same length; i.e., either six or seven letters. Thus, for six-letter words D_{56} always relates the last two letters of a word to each other; and D_{16} always relates the first character to the last. In retaining this notion of position when using word sets whose words are of varied lengths, a system loses the capability of relating only word endings to each other. That is, since D_{56} relates the last two characters of a six-letter word but the middle characters of an eleven-letter word, the system would not have an ability to interrelate only word endings. Similarly, such a system would not have an ability to interrelate the characters near the end of a word without having other interfering noise.

4.2.1. Positional n-Grams Anchored to the End of a Word

Define a backward-oriented binary diagram B_{ij} as a binary array whose elements $b_{ij}(a,b)$ are 1 if and only if the characters C_a and C_b occur in positions m+1-i and m+1-j, respectively, of some m-letter word in the dictionary. Again, we require that $i < j$ without loss of generality. In this manner, a set of these n-grams anchored from the end of words represents the endings of all words rather than a set of word endings mixed with some word middles. The definition extends to binary n-grams.

This n-gram variant permits the correlation of the letters of a word ending with the other letters of the word with less interference from word-size problems. If all of the words are of the same length, every backward-oriented n-gram of the form B_{ij} is precisely equivalent to an n-gram D_{ab} where a=m+1-j and b=m+1-i; therefore, in the context of a dictionary whose words are all of the same length, the concept of backward-oriented n-gram is a useless addition. However, let us consider a dictionary of words of different lengths but similar endings, "amicable, honorable, dependable, and denumerable." The digram D_{78} has four entries ("le," "bl," "ab," and "ra") while the backward-oriented digrams B_{12}, B_{23}, B_{34}, and others, have only one entry, indicating the strength of the relationship among the letters of the endings. Such a measure of stronger relationships would be expected to provide greater power for an error detection-correction system.

4.2.2. Positional n-Grams Anchored to Both Ends of Words

Define an ends-oriented binary digram E_{ij} as a binary array whose elements $e_{ij}(a,b)$ are 1 if and only if the characters C_a and C_b occur in positions i and m+1-j, respectively, of some m-letter word in the dictionary. The first index is the index of a position in a word counted from the beginning of the word; the last index is the index of a word position counting from the end of the word. This invention allows a system which is to use words of different lengths to interrelate the ends of words with each other. In extending the concept to binary n-grams there is an ambiguity which must be resolved. To what positions does the ends-oriented trigram E_{ijk} refer? When it is necessary to differentiate, the following notation will be used. Let E'_{ijk} be an ends-oriented

trigram which refers to positions i, j, and m+1-k while E''_{ijk} refers to positions i, m+1-j, and m+1-k. As in the case of the backward-oriented n-grams, if all of the dictionary words are of the same length there is no need for the concept of ends-oriented n-grams since every ends-oriented n-gram is precisely equivalent to some conventional n-gram.

4.2.3. Notation

Some conventions of notations must be adopted. Since the system is to work with words of different lengths it is possible to apply the digram D_{16} to a five-letter word; that is, the digram refers to a position which does not exist. Since the particular positions to which some n-grams are to be applied may not be defined in some m-letter word, define x_i = space if $i < 1$ or $i > m$. Thus, every word is considered to be padded with a sufficiently large number of blanks, left and right.

If it is necessary to differentiate among the various types of n-grams which have been defined, the positional n-grams of chapter III will be referred to as being forward-oriented n-grams. They will be abbreviated as D, regardless of their dimensions; the notations D_{ij} and D_{ijk} will be used when it is necessary to refer to particular indices.

The position function, Pos, is to be defined for the additional n-gram types as yielding the position indices of the n-gram to which it is to be applied but always counting positions from the beginning of the word; for example, $Pos(B_{ij}) = \{m+1-i, m+1-j\}$ where m is the length of the word being processed. An additional pair of position functions will be useful in the treatment which follows. Define the position functions Posf and Posb as functions which are applied only to ends-oriented n-grams. Posf can be said to yield only the set of forward-counted indices while Posb can be said to yield only the set of backward-counted indices. If m is the length of the words to which the n-grams are to be applied, then:

$$\mathrm{Posf}(E_{ij}) = \{i\},$$
$$\mathrm{Posb}(E_{ij}) = \{m+1-j\},$$
$$\mathrm{Posf}(E'_{ijk}) = \{i,j\},$$
$$\mathrm{Posb}(E'_{ijk}) = \{m+1-k\},$$
$$\mathrm{Posf}(E''_{ijk}) = \{i\}, \text{ and}$$
$$\mathrm{Posb}(E''_{ijk}) = \{m+1-j, m+1-k\}.$$

4.3. DETECTION, LOCATION, AND CORRECTION OF ERRORS

The first part of this section relates the n-gram concepts of this chapter to the error types of the preceding chapter (substitution and transposition errors). The latter part of the section shows how the backward-oriented n-grams and ends-oriented n-grams may be used to correct insertion, deletion, split, and merger errors.

4.3.1. Substitution Errors

The statements made in chapter III concerning the detection, location, and correction of substitution errors are valid for the various types of n-grams defined in this chapter; however, the notation must be updated. Given a collection of arbitrary n-grams, where the n-grams are

of various types, if X has q substitution errors, in positions $m_1, m_2, \ldots, m_q$, then

$$\text{for each } A \text{ such that } A(X)=0,$$
$$\exists i,\ 1 \le i \le q,\ m_i \in \mathrm{Pos}(A).$$

That is, regardless of the characteristics of the particular n-gram A, if A(X)=0 at least one of the m_i must be a position relevant to A, otherwise the hypothesis of exactly q substitution errors in the given positions must be false.

The method of correction of substitution errors with the additional types of n-grams is essentially the same as described in chapter III. The general technique, intersection of the appropriate rows and columns, is still used. The addition to the technique comes from the double-ended nature of the ends-oriented n-grams. For example, given an m-letter word X in which n-grams are to be used to correct an error in position r, there are three ways in which ends-oriented n-grams may refer to this position. The n-grams may be of the form $E_{i,m+1-r}$, $E_{r,j}$, and $E_{r,m+1-r}$. For example, in "BO*TON" E_{31} refers to the third and last characters of the word while E_{14} refers to the first and third characters and E_{34} refers to only the third position. An implementation must use the first two of these in the conventional manner. The row vector which is to be used for the third case is the diagonal of the array $E_{r,m+1-r}$. The strong "BO*TON" can only be corrected in the third position if there are 1's on the diagonal of E_{34}. Any admissible letter must be admissible in both positions r and m+1-r; thus, if $e_{34}(12,13)=1$ and $e_{34}(19.$ [illegible]$)=1$, both

"BOLTON" and "BOSTON" are possible corrections to "BO*TON." When using arbitrary n-grams the diagonal used must be carefully selected. For example, for six-letter words the trigram E'_{354} refers to positions 3, 5, and 3; thus, for "BO*TON," the diagonal of E'_{354} to be used is that set of elements $E'_{354}(i,x_5,i)$ for i=a to z.

In the remainder of this section, methods of detection, location, and correction of other types of errors are presented. The types of errors to be discussed are insertions and deletions and some new but similar types, splits and mergers. In this chapter we discuss methods for correcting only single occurrences of these types of errors; appendix A presents a method for location and correction of double deletion errors and the manner in which this method may be extended to locate and correct multiple deletion, insertion, merger, or split errors.

4.3.2. Deletion Errors

If $X=x_1x_2...x_m$ has a deletion error in position r, then there exists a character y and a dictionary word Y such that $Y=x_1...x_{r-1}y\ x_rx_{r+1}...x_m$. The index of the position of the error may be any number from 1 to m+1. For example, if X="SPRINGIELD," r=7, y="F," and Y="SPRINGFIELD."

The process of locating and correcting deletion errors is different from that used to locate and correct substitution errors. Correction could be blindly attempted by stretching out the word X and attempting to correct each of the resulting m+1 forms as if they were single substitution errors. This would be tedious and inefficient since it is quite uncertain that X even has a deletion error. Thus, it is desirable to be

able to determine that X has no deletion errors or at least to limit the search for corrections.

If the only error in X is a deletion error in position r, and $\mathfrak{D}$ is a collection of n-grams, then:

a) if $D(X)=0$, $D \in \mathfrak{D}$, $r \leq \max(\mathrm{Pos}(D))$,

b) if $B(X)=0$, $B \in \mathfrak{D}$, $r > \min(\mathrm{Pos}(B))$,

c) if $C'(X)=0$, $C' \in \mathfrak{D}$, $r \leq \max(\mathrm{Pos}(C'))$
<u>and</u> $r > \min(\mathrm{Pos}(C'))$,

d) if $E(X)=0$, $E \in \mathfrak{D}$, $r \leq \max(\mathrm{Posf}(E))$
<u>or</u> $r > \min(\mathrm{Posb}(E))$.

Let us consider the cases in order:

a) If $D_{ijk}(X)=0$, $r \leq k$ because $D_{ijk}(X)=0$ implies that something is wrong with the combination of characters x_i, x_j, and x_k in positions i, j, and k, respectively. Since the hypothesis is that the only error in X is a deletion error, if it is in position $r > k$, it cannot affect the admissibility of x_i, x_j, and x_k; therefore, r cannot be greater than max(Pos(D)) if D has rejected X.

b) The reasoning for the case of backward-oriented n-grams is similar to that for forward-oriented n-grams. If $B_{ijk}(X)=0$, $r > m+1-k$ because, under the assumption of a single deletion error, r must be greater than the index of the leftmost character which appears to be in the wrong location. Consider the following example. Suppose X="SPRINGIELD", $B_{145}(X)=0$ implies

$$r > \min(\mathrm{Pos}(B))$$

$$r > \min(\{6,7,10\})$$

$$r > 6.$$

This provides the information that each position in the set $\{7,8,9,10,11\}$ could contain the error.

The subtle "equality" difference between the two conditions, $r \leq \max(\mathrm{Pos}(D))$ and $r > \min(\mathrm{Pos}(D))$ should be explained since it will recur in various forms. The forward-oriented n-gram expresses the fact that the deleted character should go before the character pointed to by a forward-oriented n-gram, or possibly even in the indicated position. However, the deleted character must go after the position indicated by a backward-oriented n-gram because it cannot go into the indicated position. Note that in X="SPRINGIELD," (r=7) it is possible for $D_{147}(X)=0$ but not $D_{146}(X)=0$; also, it is possible for $B_{125}(X)=0$ but not $B_{124}(X)=0$.

c) If X has a deletion error and a non-positional trigram C'_i rejects X, the combination $x_i x_{i+1} x_{i+2}$ never appeared in adjacent positions in the dictionary. For example, if r=i+1 and $C'_i(X)=0$, $x_i x_{i+1} x_{i+2}$ violates the dictionary syntax. On the other hand, if $r \leq i$ or $r > i+2$, it would not be possible for $C'_i(X)=0$ since $x_i x_{i+1} x_{i+2}$ must appear in the dictionary. Thus, if $C'(X)=0$, convenient bounds are placed on the possible values of r; i.e., $i < r \leq i+2$. In "SPRINGIELD," the non-positional trigrams $C'_5(X)$ or $C'_6(X)$ might be zero but no other C'_i can be zero if "SPRINGFIELD" is in the dictionary. Convenient bounds are placed on the position r if any $C'(X)=0$.

d) Since the ends-oriented n-grams encompass the features of both the forward-oriented and backward-oriented n-grams, the logic is similar. However, if E(X)=0, all that is determined is that something is wrong with either a set of positions at the beginning of the word or a set of positions at the end; i.e., either the first max(posf(E)) or the last m-min(Posb(E)) positions of the word. For example, if X="SPRINGIELD", $E_{45}(X)=0$ means that no dictionary word has an I in the fourth position and a G in the sixth position (fifth position from the end); it also implies that a deletion error must be in either the first four positions of X or the last five. (In this case, the last five are: after the G, after the I, ..., and after the D.)

The algorithm for locating the position of a deletion error follows from the above. Suppose that some $D_{ijk} \in \mathcal{D}$ has rejected an m-letter word X which has one deletion error. $D_{ijk}(X)=0$ implies that the position of the error, $r \leq k$ or, equivalently, $r \in [1,k]$; that is r is a member of the interval of integers [1,k]. Similarly, $B_{ijk}(X)=0$ implies that the position of the error $r > m+1-k$ or, equivalently, $r \in (m+1-k,m+1] = [m+2-k,m+1]$, where the half-open interval of integers is more clearly represented by the equivalent closed interval of integers [m+2-k, m+1]. If $D_{ijk}(X)=0$ and $D_{ijk'}(X)=0$ then $r \leq \min(k,k')$ which is equivalent to the statement $r \in [1,k] \cap [1,k']$. In this manner,

$$r \in \Big(\bigcap_{\substack{D\in\mathcal{D}\\ D(X)=0}} [1,\max(\mathrm{Pos}(D))]\Big)$$

$$\cap \Big(\bigcap_{\substack{B\in\mathcal{B}\\ B(X)=0}} [\min(\mathrm{Pos}(B))+1,m+1]\Big)$$

$$\cap \Big(\bigcap_{\substack{C'\in\mathcal{C}\\ C'(X)=0}} [\min(\mathrm{Pos}(C'))+1,\max(\mathrm{Pos}(C'))]\Big).$$

Thus, an interval [i,j] which contains r is obtained. If the interval is empty, it cannot be true that X contains exactly one deletion error. (In the event that there are no rejecting n-grams in one or more of the above sets; the empty intersection is defined as the interval [1,m+1]. For example, if there is no $D \in \mathcal{D}$, such that $D(X)=0$, then

$$\Big(\bigcap_{\substack{D\in\mathcal{D}\\ D(X)=0}} [1,\max(\mathrm{Pos}(D))]\Big)=[1,m+1].$$

Let us now incorporate the positional information which is to be gained from the ends-oriented n-grams. Consider the ends-oriented digram E_{33} which correlates the characters in positions 3 and m-2. If $E_{33}(X)=0$ a deletion error must be in the set of positions
$[1,3] \cap (m-2,m+1]=[1,3] \cap [m-1,m+1]$. If X has five or fewer characters ($m \leq 5$), the pair of intervals degenerates to the interval [1,m+1]; i.e., this particular n-gram has provided no location information. The information which is to be gained from the set of rejecting, ends-oriented n-grams is that the set of positions which may contain the error is given by

$$\bigcap_{\substack{E\in\mathcal{E}\\ E(X)=0}} ([1,\max(\mathrm{Posf}(E))] \cup [\min(\mathrm{Posb}(E))+1,m+1])$$

which can be intersected with the previous set of positions to determine the set of indices of positions in which it is possible for a substitution error to occur. Then, $r \in S$, where

$$S = \Big(\bigcap_{\substack{D\in\mathcal{D}\\ D(X)=0}} [1,\max(\mathrm{Pos}(D))]\Big) \cap \Big(\bigcap_{\substack{B\in\mathcal{B}\\ B(X)=0}} [\min(\mathrm{Pos}(B))+1,m+1]\Big)$$

$$\cap \Big(\bigcap_{\substack{C'\in\mathcal{C}\\ C'(X)=0}} [\min(\mathrm{Pos}(C'))+1,\max(\mathrm{Pos}(C'))]\Big)$$

$$\cap \Big(\bigcap_{\substack{E\in\mathcal{E}\\ E(X)=0}} ([1,\max(\mathrm{Posf}(E))] \cup [\min(\mathrm{Posb}(E))+1,m+1])\Big).$$

To correct a deletion error, the CPP must iteratively expand the word to contain m+1 characters, leaving a space in each position $r \in S$ which might contain the deletion error. Then the CPP must determine which character can fit the space which has been left, given the context of the m letters in the rest of the word. If the error can be unambiguously corrected, only one character will fit the several insertions. As with the substitution errors in chapter III, the system could attempt to correct a suspected deletion error in all positions 1, 2, ..., m+1 without raising any additional ambiguities or erroneous corrections; the method of position location only serves to reduce the amount of searching needed to determine all possible corrections.

The ends-oriented n-grams are a special case because they have the potential to divide a region S into disjoint intervals. It has already been shown that for a word X with a deletion error in position r,

$E_{ij}(X)=0$ implies that either $r \geq i$ or $r < m+1-j$; now an example will be presented to illustrate the manner in which the interval of location may be broken into three parts. Let X be a six-letter word in which it is known that the interval S has been determined to be [2,6] before applying any knowledge of the rejecting ends-oriented n-grams. Suppose that X has been rejected by E'_{142} and E''_{243}. This information may be applied in the following manner:

$$\begin{array}{llllll} & \mathrm{Pos}(E'_{142}) = \{1,4,5\} & & \mathrm{Pos}(E''_{243}) = \{2,3,4\} \\ & \max(\mathrm{Posf}(E'_{142})) = 4 & & \max(\mathrm{Posf}(E''_{243})) = 2 \\ & \min(\mathrm{Posb}(E'_{142})) = 5 & & \min(\mathrm{Posb}(E''_{243})) = 3 \\ [2,6] \cap & ([1,4] \cup [6,7]) & \cap & ([1,2] \cup [4,7]) \\ & ([2,4] \cup [6]) & \cap & ([1,2] \cup [4,7]) \\ & & [2] \cup [4] \cup [6] & \end{array}$$

The error is in position 2, 4, or 6. The interval [2,6] was selected for the example because it is the shortest interval which can be broken into three disjoint subintervals; of course, there are many ways in which an interval may be broken into two or more parts.

4.3.3. Insertion Errors

If $X=x_1 \ldots x_m$ has an insertion error in position r, the correct spelling of X is $Y=x_1 \ldots x_{r-1}x_{r+1} \ldots x_m$ for some dictionary word Y. The error position r may be any position from 1 to m. It will be found to satisfy the following inequalities, similar to those given in the preceding

section for deletion errors.

a) if $D(X)=0$, $D \in \mathcal{E}$, $r \leq \max(\mathrm{Pos}(D))$,

b) if $B(X)=0$ $B \in \mathcal{E}$, $r \geq \min(\mathrm{Pos}(B))$,

c) if $C'(X)=0$, $C' \in \mathcal{E}$, $r \leq \max(\mathrm{Pos}(C'))$ and $r \geq \min(\mathrm{Pos}(C'))$,

d) if $E(X)=0$, $E \in \mathcal{E}$, $r \leq \max(\mathrm{Posf}(E))$ or $r \geq \min(\mathrm{Posb}(E))$.

Let us consider the cases in order:

a) If $D_{ijk}(X)=0$, $r \leq k$ because $D_{ijk}(X)=0$ implies that the combination of characters x_i, x_j, and x_k is inadmissible in positions i, j, and k, respectively. Since the hypothesis is that the only error in X is an insertion error in position r, if $r > k$, an insertion error cannot possibly affect the admissibility of $D_{ijk}(X)$. Thus, r cannot be greater than max(Pos(D)) if D has rejected X.

b) The reasoning for a backward-oriented n-gram is similar to that for a forward-oriented n-gram. If $B_{ijk}(X)=0$, $r \geq m+1-k$ since under the assumption of a single insertion error, r must be in or after the position m+1-k. That is, an insertion error in position $r < m+1-k$ cannot affect the admissibility of $B_{ijk}(X)$; therefore, $r \geq m+1-k$. Consider the following example. Suppose X="SPRINGEFIELD"; $B146(X)=0$ implies

$$\begin{aligned} r &\geq \min(\mathrm{Pos}(B)) \\ r &\geq \min(\ 7,9,12\ ) \\ r &\geq 7. \end{aligned}$$

This provides the information that each position in the set $\{7,8,9,10,11,12\}$ could contain the error.

c) If X has an insertion error and a non-positional trigram C'_i rejects X, the combination $x_i x_{i+1} x_{i+2}$ never appeared in adjacent positions in the dictionary and is inadmissible. On the other hand, if $r < i$ or $r > i+2$, it would not be possible for $C'_i(X)=0$ since $x_i x_{i+1} x_{i+2}$ must appear in the dictionary.

d) Since the ends-oriented n-grams encompass the features of both the forward-oriented and backward-oriented n-grams, the logic is similar. However, if $E(X)=0$, all that is determined is that something is wrong with either a set of positions at the beginning of the word or a set of positions at the end; i.e., either the first $\max(\text{Posf}(E))$ or the last $m-\min(\text{Posb}(E))+1$ positions of the word. For example, if X="SPRINGEFIELD," $E_{46}(X)=0$ implies that an insertion error must be in either the first four positions of X or the last six.

The algorithm for locating the position of an insertion error is similar to the algorithm for detection and location of a deletion error. The error position r is known to be in the set S where

$$S = \Big(\bigcap_{\substack{D \in \mathcal{D} \\ D(X)=0}} [1,\max(\text{Pos}(D))] \Big) \cap \Big(\bigcap_{\substack{B \in \mathcal{B} \\ B(X)=0}} [\min(\text{Pos}(B)),m] \Big)$$

$$\cap \Big(\bigcap_{\substack{C' \in \mathcal{C}' \\ C'(X)=0}} [\min(\text{Pos}(C')),\max(\text{Pos}(C'))] \Big)$$

$$\cap \Big(\bigcap_{\substack{E \in \mathcal{E} \\ E(X)=0}} \big([1,\max(\text{Posf}(E))] \cup [\min(\text{Posb}(E)),m] \big) \Big).$$

To correct an insertion error, the CPP must iteratively shorten the word to contain m-1 characters, deleting the character in each position $r \in S$ which might contain the insertion error. Then the CPP must determine whether or not the resulting word is in the dictionary. As with the substitution errors in chapter III, the system could attempt to correct insertion errors in all positions 1, 2, ..., m without raising any additional ambiguities or erroneous corrections; the method of position location only serves to reduce the amount of searching needed to determine all possible corrections.

4.3.3.1. An Example of Correction of an Insertion Error

Figure 4.1 presents an example which illustrates the ability of the additional n-gram types to limit the search time required for correcting an insertion error. The forward-oriented binary digram D_{15} indicates that, if there is an insertion error in the word "FEARDED," it must be in or before position five. Without additional algorithms, five corrections would have to be attempted in order to assure that the word is unambiguously corrected. Application of backward-oriented n-grams further reduces the number of searches required to three since they indicate that the error is in or after position three. The ends-oriented n-gram E_{13} further reduces the possibilities to position five. When the letter in position five of "FEARDED" is deleted, the resulting word "FEARED" is found to be in the dictionary. Non-positional binary n-grams provide no assistance in this sample.

The Dictionary:	HEAD
	BREAD
	BREADED
	HEARD
	FEARED
	HERDED

Input word: FEARDED

$d_{15}(F,D) = 0 \Rightarrow r \leq 5.$

$\Rightarrow$ the correct word may be either

EARDED, FARDED, FERDED, FEADED, or FEARED.

$b_{15}(D,A) = 0$, $b_{25}(E,A) = 0$, and $b_{35}(D,A) = 0 \Rightarrow r \geq 3.$

$\Rightarrow$ the correct word may be either

FERDED, FEADED, or FEARED, when used with the information from the forward-oriented n-grams.

$e_{13}(F,D) = 0 \Rightarrow r \leq 1$ or $r \geq 5.$

$\Rightarrow$ the correct word must be FEARED, if FEARDED contains an insertion error.

Figure 4.1 An example of the use of various types of n-grams to locate the position of an assumed insertion error.

Again it must be pointed out that the example is simple-minded and only illustrative. In actual problems, such as those explored in the next chapter, a number of binary trigrams of various types is used; and much larger dictionaries are used as data bases.

4.3.4. Mergers and Splits

Although spelling errors are primarily substitutions, insertions and deletions (Masters, 1927), the CPP which works with OCR equipment should also allow for correction of machine generated segmentation errors. Such segmentation errors stem from an inability to properly delineate one or more characters. Define a merger error as the joining and classification of two distinct characters as a single character. This is not the same as a deletion error since the resulting joined character generally would be classified as a character different from either of the original joined characters. Define a split error as the improper segmentation and classification of a single character as a pair of characters. Again, this is different from an insertion because it is most probable that neither of the resulting classified characters is the same as the original character. A split in which one of the resulting characters is given the label of the original character may be treated as an insertion; and a merger in which the merged character is labelled the same as one of the original characters may be treated as a deletion.

These errors were referred to as machine generated segmentation errors because they are caused by the inability of a pattern recognition machine to properly segment strings of characters in the manner in which the author intended. However, the segmentation problem is usually caused by flaws in the source document such as broken type and poor or crowded printing or handwriting. Humans are quite capable of making these segmentation errors also.

4.3.4.1. Merger Errors

If $X=x_1x_2...x_m$ has a merger error in position r, there exist characters y and y' and a dictionary word Y such that $Y=x_1x_2...x_{r-1}\ y\ y'\ x_{r+1}...x_m$ is the correct spelling of X.

Determination of the location of a merger error is identical to that process for an insertion error. The reason for this is that the erroneous character x_r does not belong in the r-th position of the word. Although the insertion involves the presence of an erroneous character and the merger implies the absence of two characters with the substitution of another in their place, both problems have the same symptoms; the character in position r is incorrect. For example, in the merger X="SPRINGXELD" (r=7), the character "X" must be replaced by "FI" and in the insertion X="SPRINGYFIELD" (r=7), the character "Y" must be removed; both cases have an error in the seventh position and both display the same symptoms to an error detection and location system.

After determining the set S in which an insertion error could occur, the CPP can use the same set as the set of positions in which a merger error could occur. A merger error is corrected by first expanding the word to m+1 letters by inserting a space into each potentially erroneous

position and then treating the resulting word as if it had adjacent substitution errors in positions r and r+1. For example, if X="SPRINGXELD" and S=[6,8], try correcting "SPRIN*GXELD," "SPRING*XELD," and "SPRINGX*ELD."

4.3.4.2. Split Errors

If $X-x_1x_2...x_m$ has a split error in position r, there exist a character y and a dictionary word Y such that $Y=x_1x_2...x_{r-1}\ y\ x_{r+2}...x_m$ is the correct spelling of X. If the only error in X is a split error in position r and is a collection of n-grams, then:

a) if $D(X)=0$, $D \varepsilon \mathcal{P}$, $r \leq \max(Pos(D))$,

b) if $B(X)=0$, $B \varepsilon \mathcal{P}$, $r \geq \min(Pos(B))-1$,

c) if $C'(X)=0$, $C' \varepsilon \mathcal{P}$, $r \leq \max(Pos(C'))$
<u>and</u> $r \geq \min(Pos(C'))-1$,

d) if $E(X)=0$, $E \varepsilon \mathcal{P}$, $r \leq \max(Posf(E))$
<u>or</u> $r \geq \min(Posb(E))-1$.

Determination of the position of the error differs from the other three forms because of a subtle difference in the definitions. When locating deletion errors, intervals of the form [min(Pos(B))+1,m+1] arose because the character which appeared to be in the wrong location (min(Pos(B))) actually had to precede the missing character and was actually in its proper location. When locating insertions and mergers, the intervals were of the form [min(Pos(B)),m] because the errant character could have been in the position which appeared to have been in error. Now, however, intervals of the form [min(Pos(B))-1,m] are encountered because, while the character observed to be out of place may be in error (one of the two "split-ees"), the error position may be the position prior to the apparent position; that is, location r+1 may be observed to be in error because it is only part of the error.

4.4. SUMMARY AND DISCUSSION

In this chapter, new developments in the method of binary n-grams have been presented. The concept of binary n-gram has been extended to include notions of positional anchoring. It was shown that an n-gram could be considered to interrelate only word endings, word beginnings, or both ends of words. It was also noted that the additional concepts are of no use unless the dictionary is to contain words of different lengths.

The algorithms for error detection and correction were extended to include the classical spelling errors, insertions and deletions. Methods of detection, location, and correction were presented for two additional types of errors. These errors, splits and mergers, are essentially machine generated errors. It was shown that while the errors appear to be similar both the source of the errors is different from the source of insertion and deletion errors and the algorithms for correction are slightly different from the algorithms to be used for correcting insertions and deletions.

In the next chapter, experimental results will be presented which attempt to determine the relative effectiveness and value of the methods developed in this chapter. The experimental results will examine the

effectiveness of the CPP algorithms for different sizes of dictionaries, different quantities of CPP storage, and different mixes of n-grams. One of the implications of these results is that, given a fixed quantity of storage, a CPP which divides the storage among different types of n-grams is more effective than a CPP which uses only one type of n-gram.

No attempt was made in this chapter to discuss any methods for correction of multiple deletion, insertion, etc., errors. For a classifier error rate of 10%, the expected proportions of six-letter words with one and two errors are 39.9 and 17.6%, respectively, a ratio of 2.26:1. For lower rates the proportion of single to double error words would be even higher. The computation required for correction of double errors is about four times that required for single errors; triple errors require about four times as much effort as double errors--we do not attempt correction of triple errors in this research. On the other hand, however, is the fact that so much more computation must be done to provide just a little bit greater correction. At some point correction attempts must cease. If the recognition rate is 97% and triple errors represent .5% of the data, we know that correcting all of them will only raise correction to 97.5%; an analysis must determine whether or not that .5% is worth it. In the research of Hanson, et al., (1975) a 96.4% word recognition rate with 2.3% error was achieved; the remaining percentage belonged to the triple error words. Correction of the triple error words raised recognition to 97.3%.

We are now faced with a problem of even greater complexity. Five different types of errors have been considered; each of these requires a slightly different algorithm for location and/or correction. There are 25 ways in which they may occur in combinations to make a two error word. We feel confident that correction would be just as effective for most of these as it is for double substitution errors; however, no real world distribution data exists to show whether or not the various combinations occur often enough for the algorithms to be worthwhile. As an example of the location and correction methods which may be used for these double errors, appendix A describes a method for locating and correcting double deletion errors.

C H A P T E R V
EXPERIMENTAL RESULTS

5.1. INTRODUCTION

The preceding two chapters detailed the methods of binary n-grams: chapter III reviewed the prior research and chapter IV presented some new techniques and related the new techniques to detection and correction of additional types of errors. To examine the effectiveness of the algorithms, some experiments were performed on a fixed CPP design which uses several algorithms and various dictionary sizes.

Experiments were then performed to examine the effectiveness of the same CPP when larger and smaller contextual data bases are employed. It is also shown that CPP effectiveness is related to the length of the words being processed.

In the above experiments, various different mixtures of n-gram types were used. Because of the expense of finding good sets of n-grams, only a few such experiments were performed. Section 5.4 briefly describes the preliminary experiments which led to the use of the particular sets of n-grams used in the experiments of this chapter. The important result is that little is gained from either the non-positional quadgram or the ends-oriented n-grams until a very large contextual data base is required.

5.2. BACKGROUND OF THE EXPERIMENTS

In this section, the groundwork is laid for the experiments to be performed. First, the experimental design of the CPP is presented. Then, the source of the "dictionary" words is presented along with a brief description of the words in the dictionary.

5.2.1. The Dictionary

The master dictionary for all of these experiments was obtained by extracting the city (or town) names for all of the first, second, third, and fourth class post offices in the U.S.A. from the Zip-A-List tapes (USPS, 1973). Spaces were deleted from the names. Names with digits and names longer than 16 characters were deleted from the list. Af er duplicates were also deleted, 19196 names remained. It must be noted that these names are not typical of English text; a statistical comparison of the word list to other lists of typical words is presented in Appendix B.

In order to compare the effectiveness of the CPP design when applied to dictionaries of different sizes, three subdictionaries were created. The names were selected for each dictionary so that it is also a subdictionary of the next larger sized dictionary. Thus, 1000 names were selected for the smallest dictionary; an additional 4000 were selected for the dictionary of 5000; and 5000 more were selected for the dictionary of 10000. To assure that the smallest dictionary has an adequate number of words of each length, the selection procedure was biased to guarantee that each length set has at least 20 names. (Table B.1 in Appendix B presents the distributions of the word lengths in the four dictionaries.)

5.2.2. The Test Data

Several types of errors were generated in each of the words in the 1000 word dictionary. The generated strings were then used in the experiments to be described in the next section. Thus, even though the contextual data base of an experiment might be the 10000 word dictionary, the CPP design would be tested with the same set of data which was used to test each of the other CPP designs.

With a few exceptions the results shown in this chapter will describe the capability of the CPP only as it applies to single substitution errors, insertion errors, and double substitution errors: substitutions were used because they provide the best link with prior research and insertions because they demonstrate the effectiveness of the new algorithms. Lest we be accused of sweeping everything else under the rug, the first experiment will present the results of processing several types of errors. Errors were generated in each of the test words by first randomly generating the position of the error and then the error.

There were two basic types of errors generated. For the first type of error, letters were selected randomly to fill the position of the error. For the second type, a reject character ("*") was chosen to fill the questionable position. The reason for the two types is that they provide a comparison between a classifier which makes a guess when it cannot decide the identity of a character and a classifier which rejects the input datum rather than make a mistake. These procedures were used in almost all cases; one exception is the deletion errors which are unrelated to such a methodology. A complete set of 1000 of each of the following types of errors was generated; again, these were all used for the experiment of section 5.3.1, but only three types were used for the subsequent experiments.

One substitution--A character, different from the erred character is randomly selected to fill the position of the error.

One substitution, rejected--A rejected character is put into the erred position.

One deletion--The word is shortened to omit the character in the selected position.

One merger--An error character is randomly generated, different from the two relevant characters; it is then substituted in their place and the word is shortened.

One merger, rejected--The two relevant letters are replaced by a reject character and the word is shortened.

One insertion--A letter is randomly selected and inserted into the erred position.

One insertion, rejected--A reject character is inserted into the erred position.

One split--A pair of characters, both different from the erred character, is generated to replace the erred letter.

One split, rejected--A pair of reject characters replace the erred character.

Two substitutions--Two randomly selected characters replace the erred characters.

Three substitutions--Three randomly selected characters replace the erred characters.

5.2.3 Experimental Design of the CPP

5.2.3.1. Contextual Data Base

For a majority of these experiments, the CPP will use a simple mixture of n-gram types: some forward-oriented n-grams and some backward-oriented n-grams. Some of the experiments will also use some ends-oriented n-grams and a non-positional quadgram. Section 5.4 outlines the actual experiments which selected the mixtures of n-grams.

5.2.3.2. The Internal Decision Logic

When a string is input to the CPP, it will be first assumed that there is no error in the string. If this is found to be incorrect, i.e., some n-gram rejects the string, it will be assumed that the word has one substitution error. If this hypothesis is found to be incorrect--because there is no position in which a substitution error could occur--it will be assumed that the word has either a single deletion, insertion, merger, or split error. If no correction of this sort of error can be made, the word will be assumed to have two substitution errors. If this hypothesis is also proven to be false, the word will be rejected. The order is left fixed for all of these experiments; however, if a data model existed to show that the error types could be ranked differently, the order could be rearranged. Also, in this design, there is no inherent order among the deletion-insertion-merger-split group; if an error from this group is corrected, it is because there is no other correction possible among this group. If a word containing, say, a deletion error is rejected, the reason is either that there are several possible corrections to it as a deletion error or that there are some admissible corrections as a deletion and some as some other type of error in the group. If data existed which proved that some one of the particular errors in this group should be corrected before the others, the decision hierarchy could be altered to implement such an ordering of error types.

5.3. EXPERIMENTS

The first experiment, in section 5.3.1, demonstrates the effectiveness of a single CPP design against all of the types of errors which were generated. This is done to give the reader a feel for the power of the CPP when operating upon the various types of errors; such detailed results are not again presented in this chapter. In the experiments in section 5.3.2, the performance of the CPP is compared for the four different dictionary sizes; for three types of errors (single substitutions, insertions, and double substitutions) the relationship between dictionary size and CPP effectiveness is presented. In section 5.3.3, a figure is presented which shows the relationship between the CPP's ability to detect and correct errors and the length of the words which contain those errors.

The last set of experiments (5.3.4) examines the effectiveness of the CPP as the contextual data base varies in size. In two experiments, th CPP is given first a smaller number of n-grams with which to process text and then a much larger number of n-grams. These results are then compared to those obtained by using a similar CPP which uses a dictionary before deciding whether or not a string is admissible.

5.3.1. As a Function of the Various Types of Errors

In these experiments, the CPP's contextual data base will consist of seventy forward-oriented n-grams and seventy backward-oriented n-grams. The dictionary of 10000 words is used.

The results of testing the CPP against all of the various types of errors are presented in table 5.1. Of the 1000 single substitution errors, 98.6% were detected and 81.0% were corrected; there were only 1.4% in error. Because of the CPP design, all of the 14 CPP errors were words in which the CPP did not detect the substitution error. Note that the second line of table 5.1 is better than the first; it provides the results of processing words in which the simulated classifier rejected a letter instead of making a bad guess. It is better because reject characters ("*") are always detected and because the position of the error is immediately determined. (For example, "B*STON" is easily corrected as "BOSTON" but "BASTON" could also be corrected to "GASTON" since the position of the error cannot be unambiguously determined.)

TABLE 5.1

EFFECTIVENESS OF A FIXED CPP DESIGN WHEN APPLIED TO SEVERAL ERROR TYPES

Error type	Detected as having appropriate error type	Corrected	Rejected		Errors	
			Detected & Rejected	Rejected as different type of error	Undetected	Improperly modified
One substitution. .	986	810	176	—(a)	14	—
One rejection . . .	1000	840	160	—	—	—
One deletion. . . .	760	577	183	143	33	64
One merger.	653	520	277	104	11	66
One merger,rejected	873	596	277	78	—	49
One insertion . . .	958	929	29	19	1	22
One insertion,rej'd	982	967	15	10	—	8
One split	971	812	159	10	1	18
One split,rejected.	1000	842	156	—	—	—
Two substitutions .	848	632	216	77	1	73
Three substitutions	827	—	827	100	1	71

Note: Dictionary size, 10000 strings,
Storage requirement 2.8 million bits.

(a) a dash (——) is used to indicate that the particular item is not applicable.

Deletions and mergers form a weak spot in contextual processing. Of the 1000 deletion errors processed, only 760 are detected as having at least a deletion error and only 577 of them are corrected. Of the remaining 240 which were not detected as having deletion errors, 33 had no detectable error and the other 207 could have been one error words; 64 of these were unambiguously modified as if they had been single substitution errors and the remaining 143 were rejected. In summary, the CPP made a total of 577 corrections and 97 errors.

Insertion and split errors are detected and corrected even better than substitution errors. Again, slightly better detection and correction rates are found when rejections are output by the simulated classifier instead of erroneous classifications. In the case of merger errors this improvement is quite marked: the rejection flags the position of the error so well that 22% more of the words in error are detected, 7.6% more are corrected, and there are half as many errors. In the case of insertion errors, a rejection corresponds more closely to the machine classification of a smudge or other noise on the input document than to the usual insertion which is a spelling error.

More than half of the double substitution errors are corrected. It should be noted that of the 152 strings in which the double substitution error was not detected, 103 were presumed to have been of the insertion-deletion-merger-split variety and that 73 of these were erroneously corrected. If the data were to indicate that the system should be redesigned, these 103 strings could also have been detected as having double substitution errors. That is, if experimentation were to indicate that it might be profitable, we might examine all (or most) possibilities before committing the CPP to make a decision. Another redesign would place the insertion-deletion-merger-split decision after the double substitution error decision and no double substitution error words would be mistakenly identified as insertions, deletions, etc. Of course, with such a redesign, some of the words with deletions, insertions, etc., would then be mistakenly identified as having double substitution errors.

5.3.2. As a Function of Dictionary Size

In these experiments the effectiveness of the CPP is tested with each of the four dictionaries as a data base. The same CPP design is used in each of the four experiments, the same number of n-grams and the same order of correction. As may be seen in table 5.2, the CPP is very effective for the dictionaries of 1000 and 5000 words. There is quite a drop in effectiveness when the data base is expanded to 10000 words and another sharp drop when the dictionary of 19196 names is used.

While it is clear that the CPP's power erodes as the dictionary enlarges, the fact that, even in the worst case, the CPP is still able to detect and correct more than 40% of the words with double substitution errors is significant. Many of the corrected double substitution errors are difficult for humans to correct (Hanson, et al., 1975). In each case, the remaining double substitution error strings which are rejected have an average of less than six alternate strings. Thus, if

TABLE 5.2

EFFECTIVENESS OF THE CPP AS A FUNCTION OF DICTIONARY SIZE

Error type	Detected	Corrected	Rejected	Errors
The 1000 word dictionary				
One substitution	1000	985	15	0
One insertion	999	994	5	1
Two substitutions	981	928	56	16
The 5000 word dictionary				
One substitution	994	910	84	6
One insertion	976	973	20	7
Two substitutions	922	768	182	50
The 10000 word dictionary				
One substitution	986	810	176	14
One insertion	958	929	48	23
Two substitutions	848	623	293	74
The 19196 word dictionary				
One substitution	970	633	337	30
One insertion	887	849	119	32
Two substitutions	756	443	464	93

Note: Storage requirements, 2.8 million bits.

an additional processor were to be built to resolve ambiguities, it would have to process an average of six strings per decision on words with double-substitution errors. Hanson, et al., (1975) show that, for triple errors, such methods are effective against even 40 or more alternates; as many as 89% of the cases were corrected with an average of 22.6 alternates per case.

5.3.3. As a Function of Word Length

The outputs of the CPP were recorded for each different word length. The results are presented in Figure 5.1 for the three particular error types. In all cases the CPP performs better on long words than on short words.

It appears that the CPP has trouble with words which are longer than ten characters since each of the graphs shows one or two drops after this point. This anomaly is probably explained by the fact that the n-grams used only reach a distance of ten letters from either end of words; that is, the maximum subscript on the n-grams is 10. Thus, n-grams of both types are used to correct errors in any position of any word if it is not more than ten letters long; however, as words become longer, the letters at the beginning are out of reach of the backward-oriented n-grams and the letters at the end are out of reach of the forward oriented n-grams. This behavior is accepted since altering it without commensurately increasing the total number of n-grams in use

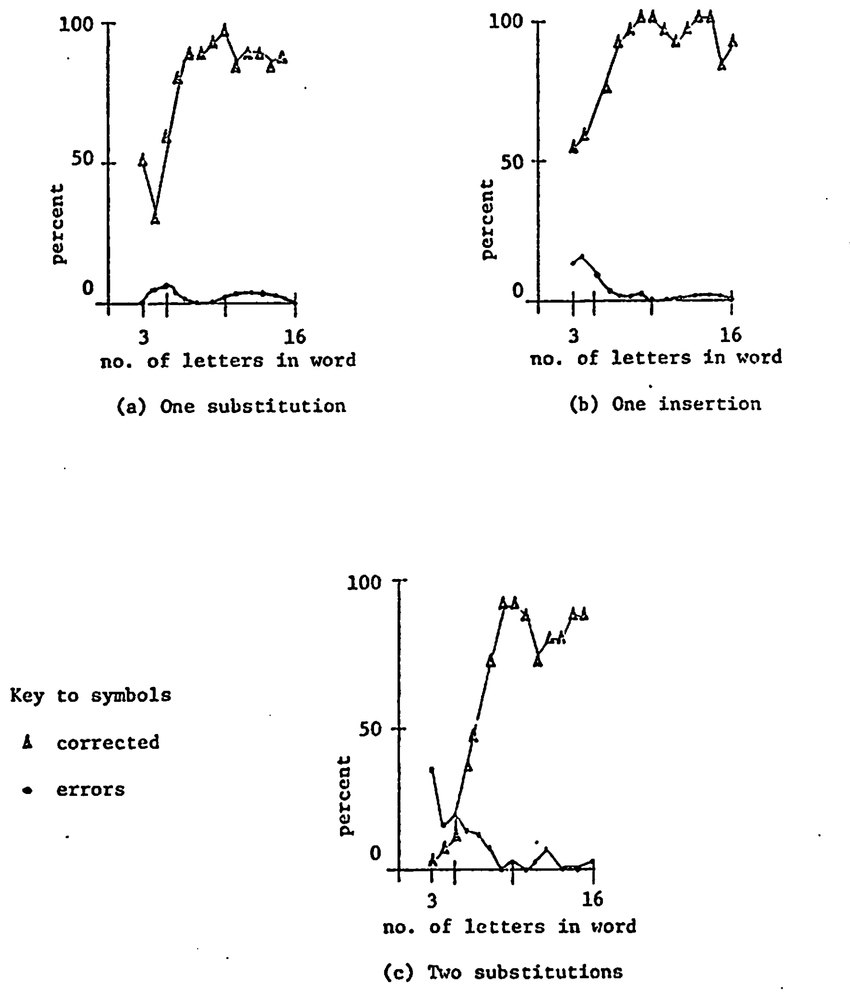

Figure 5.1 The relationship between word length and error correction and word length and errors made for three types of errors.

would affect the other correction rates; 11% of the twelve-letter words is only 0.6% of the data set while 5.0% of the nine-letter words is also 0.6% of the data. Thus it would not pay to improve recognition of the very long words at the expense of the intermediate length words.

5.3.4. As a Function of the Size of the Contextual Data Base

The total number of n-grams used by the CPP was altered to determine the relationship between recognition rates and storage required. In the larger CPP design, a mixture of n-grams was used: 76 forward-oriented n-grams, 76 backward-oriented n-grams, a non-positional quadgram, and 60 ends-oriented n-grams; the experiments which support this decision are described in the next section. The alternate CPP designs are, respectively, 65% smaller and 65% larger than the "standard" CPP used for the earlier experiments of this chapter. An additional experiment performed in this section was to incorporate a dictionary processor into the CPP strategy; in this case the CPP looks up each hypothesized admissible sequence in the dictionary before making a final decision concerning admissibility.

These experiments, shown in table 5.3, clearly indicate the expected result that greater quantities of storage provide better results. However, the results also indicate that the law of diminishing returns enters into play here because growth from a contextual data base of 1.8 million bits to 2.8 million bits raises correction rates about

TABLE 5.3

EFFECTIVENESS OF THE CPP AS A FUNCTION OF THE SIZE OF THE CONTEXTUAL DATA BASE

Error type	Detected	Corrected	Rejected	Errors
1.8 million bits				
One substitution	980	719	251	20
One insertion	944	889	82	29
Two substitutions	818	490	422	88
2.8 million bits				
One substitution	986	810	176	14
One insertion	944	929	48	23
Two substitutions	849	632	293	76
4.8 million bits				
One substitution	992	864	128	8
One insertion	967	952	27	21
Two substitutions	870	694	241	65
The Dictionary (at least .8 million bits)				
One substitution	999	945	54	1
One insertion	980	968	15	17
Two substitutions	903	783	144	73

Note: Dictionary size, 10000 strings.

10% while the next step, from 2.8 to 4.8 million bits, only improves the correction rates around 6%. Note that using a dictionary of half the size (the 5000 word dictionary in table 5.2) results in a greater improvement than doubling the storage requirement of the system (table 5.3).

Appending the dictionary provides even better performance. In chapter II several subdictionary selection algorithms were presented. The methods of binary n-grams could be viewed as being a subdictionary selection algorithm. Assuming that the error in a word is properly recognized, the difference is that the previous algorithms select a list of candidate words from the dictionary; several of the words in the list are possible corrections and one of the words in the list is the correct spelling of the word. Although the n-gram methods also provide a list of words, each word in the list is an admissible correction of the questioned word but not all words in the list are in the dictionary.

The dictionary is useful in another respect in the contextual post-processor. When the dictionary is used to check alternate choices and corrections, it may find the hypothesis concerning error type to be erroneous. Suppose a two error word is input. Further suppose that the n-gram system determines that it is not admissible but that it has one substitution error and, as such, there are k admissible sequences which are potential corrections. If k=1, the n-gram algorithms would consider the word to be corrected; if k>1, the word is rejected. If, however, a dictionary is incorporated into the system and none of the k admissible

sequences is an admissible d-sequence, the CPP must reject its hypothesis of one error and attempt another hypothesis.

It is important to note that the particular n-gram system to which the dictionary system is appended does not affect the effectiveness of the dictionary subsystem. However, a difference does arise in the amount of computation time required. If the n-gram system can be implemented so that entire rows or columns of an n-gram can be accessed and processed in a single machine instruction--or rows of several n-grams in parallel--the n-gram operations would be faster than dictionary references. In this manner, use of additional n-grams would result in a smaller number of dictionary references. Such an implementation would have to consider the cost-performance curves of these components. In the work of Hanson, et al., (1975) where additional postprocessing was used to resolve ambiguities output by the CPP, the use of a dictionary to reduce the overall number of ambiguities to be processed improved CPP performance by reducing a word error rate of 2.7% to 1.7%, a 35% reduction in the number of errors.

5.4. SELECTION OF n-GRAM SETS

This section describes the experiments which selected the sets of n-grams which were used in the experiments of the preceding section. The problem is to select the best set of n-grams, given some storage size. Ideally the set should be very good on all types of errors.

However, no exhaustive search is desired; in fact, a restricted search is necessary. The most thorough of these searches examined sets of n-grams which required about 2.8 million bits of storage. These experiments attempted to evaluate n-gram strategies instead of particular n-grams; for example, one set of n-grams evaluated had 140 forward-oriented n-grams, another had 36 forward-oriented, 36 backward-oriented, and 40 ends-oriented n-grams and a non-positional quadgram.

Table 5.4 portrays the results of the experiments for a contextual data base of 2.8 million bits. Three of the collections of n-grams provide almost the same "best" correction rates; the set of 70 forward-oriented n-grams and 70 backward-oriented n-grams was selected for use because of the homogeneity of the algorithms used. From these experiments, it appears that there is nothing to be gained by using either the non-positional quadgram or the ends-oriented ngrams. An implementation which requires only two types of n-grams is simpler than any which requires three; and, in the particular case of these two types, they could both be handled by the same reentrant code. It is clear that neither the forward-oriented n-grams nor the backward-oriented n-gram provides a sufficient data base individually. (For this experiment, n-grams were assembled which had indices up to 16.)

Table 5.5 presents the results of similar, but less thorough, experiments to produce a collection of n-grams for a contextual data base of 1.8 million bits. In this case, no attempt was made to evaluate

TABLE 5.4

EFFECTIVENESS OF DIFFERENT CPP DESIGNS
WITH A TOTAL STORAGE REQUIREMENT OF 2.8 MILLION BITS

Error type	Detected	Corrected	Rejected	Errors
140 forward-oriented n-grams				
One substitution	970	630	338	30
One insertion	903	831	131	40
Two substitutions	779	446	479	75
140 backward-oriented n-grams				
One substitution	973	703	270	27
One insertion	918	880	79	41
Two substitutions	797	503	413	84
36 forward-oriented n-grams 36 backward-oriented n-grams 40 ends-oriented n-grams 1 non-positional quadgram				
One substitution	988	776	212	12
One insertion	949	909	60	31
Two substitutions	843	573	347	80
50 forward-oriented n-grams 50 backward-oriented n-grams 40 ends-oriented n-grams				
One substitution	982	812	170	18
One insertion	956	925	52	23
Two substitutions	854	614	316	70
56 forward-oriented n-grams 56 backward-oriented n-grams 1 non-positional quadgram				
One substitution	988	813	175	12
One insertion	958	931	43	26
Two substitutions	855	624	303	73
70 forward-oriented n-grams 70 backward-oriented n-grams				
One substitution	986	810	176	14
One insertion	958	929	48	23
Two substitutions	849	632	294	74

Note: Dictionary size, 10000 strings.

TABLE 5.5

EFFECTIVENESS OF DIFFERENT CPP DESIGNS
WITH A TOTAL STORAGE REQUIREMENT
OF 1.8 MILLION BITS

Error type	Detected	Corrected	Rejected	Errors
25 forward-oriented n-grams 25 backward-oriented n-grams 16 ends-oriented n-grams 1 non-positional quadgram				
One substitution	973	668	305	27
One insertion	920	835	119	46
Two substitutions	801	421	479	100
36 forward-oriented n-grams 36 backward-oriented n-grams 20 ends-oriented n-grams				
One substitution	971	720	251	29
One insertion	911	885	85	30
Two substitutions	814	492	412	96
33 forward-oriented n-grams 33 backward-oriented n-grams 1 non-positional quadgram				
One substitution	974	671	283	26
One insertion	937	8_6	96	38
Two substitutions	802	425	561	102
46 forward-oriented n-grams 46 backward-oriented n-grams				
One substitution	980	719	261	20
One insertion	944	889	82	29
Two substitutions	818	490	422	88

Note: Dictionary size, 10000 strings.

sets of only forward-oriented n-grams or only backward-oriented n-grams. In these experiments, the CPP designs which used the non-positional quadgram were clearly inferior to the other two and the design which used the forward-oriented and backward-oriented n-grams was slightly better than that which used these and the ends-oriented n-grams. For the same reasons used above, the set of 46 forward-oriented and 46 backward-oriented n-grams was selected for use.

Little experimentation was done to select an n-gram set for the 4.8 million bit contextual data base because of the great expense of running programs this large. However, table 5.6 shows an interesting phenomenon which occurs at this level. The combination of only forward and backward oriented n-grams (120 each) is barely better than the similar mixture (70 each) for the 2.8 million bit contextual data base. In this case the mixture of n-gram types is clearly better. This may be interpreted as meaning that the contextual information which can be obtained from forward and backward oriented n-grams was almost entirely gleaned from the first set of n-grams in the smaller contextual data base. In order to extract further contextual information, the other types of n-grams, ends-oriented n-grams and a non-positional quadgram, must be employed.

TABLE 5.6

EFFECTIVENESS OF DIFFERENT CPP DESIGNS
WITH A TOTAL STORAGE REQUIREMENT
OF 4.8 MILLION BITS

Error type	Detected	Corrected	Rejected	Errors
76 forward-oriented n-grams 76 backward-oriented n-grams 60 ends-oriented n-grams 1 non-positional quadgram				
One substitution	992	864	128	8
One insertion	967	952	27	21
Two substitutions	870	694	241	65
120 forward-oriented n-grams 120 backward-oriented n-grams				
One substitution	987	823	164	13
One insertion	961	936	42	22
Two substitutions	852	663	265	72

Note: Dictionary size, 10000 strings.

5.4.1. Better Results at Lower Cost

Table 5.4 clearly indicates that, for the fixed storage size of 2.8 million bits, dividing the storage equally among both the forward-oriented n-grams and the backward-oriented n-grams (70 each) obtains better results than those obtained by using either n-gram type exclusively. Table 5.6

indicates that a much larger collection of 120 of each of these n-grams hardly does any better than the 70 each. From this an important point may be made. Given an initial CPP which uses only forward-oriented n-grams, it is more useful to discard some of them in favor of some backward-oriented n-grams than to add many more forward-oriented n-grams. It is more useful because the same, improved results with less computation and storage are obtained at a far lower cost than the results obtained by just adding backward-oriented n-grams.

5.5. SUMMARY AND DISCUSSION

This chapter has demonstrated that n-grams provide a powerful means of applying contextual information in the processing of text. Certain conjectures which would have been presumed true have been affirmed; the effectiveness of the CPP is inversely related to the size of the dictionary and directly related to both word length and number of n-grams used.

The experiments did seem to indicate the relative effectiveness of some of the n-gram components. Only for the largest of the contextual data bases is it apparent that the ends-oriented n-grams and the non-positional n-grams are effective. That is, for contextual data bases of n-grams requiring up to 2.8 million bits, it is apparent that the forward-oriented and backward-oriented n-grams adequately represent the dictionary, at least as much as any comparable volume of n-grams can represent the dictionary; however, expansion beyond 2.8 million bits with only these n-grams is useless. This would appear to indicate that the forward- and backward-oriented n-grams reach a plateau for the type of information which they provide and that additional contextual information must be provided from other sources such as the ends-oriented n-grams.

The effectiveness of any contextual system depends heavily upon the data which is input to it. To extrapolate our results of application of this system to a mix of error types requires an estimate of several parameters, the likelihoods of each error type. We can perform a simple extrapolation by assuming (a) a single error probability of 4% (independent of type, (b) that the words have the same distribution as is found in the list of 10000 words, and (c) that all words have equal a priori probabilities. In this manner. the percentage of words expected to be in error is 28.32%. Similarly, there will be only one error in 24.94% of the words. Because longer words are more likely to have an error in them than short words, 85% of the single substitution error words should be corrected; 93.5% of the single insertions and 63% of the deletion errors. If the errors are divided uniformly among these three groups, 80% of the 24.94% error should be corrected, raising the 71.68% word recognition to 91.63%.

It is expensive to evaluate these systems, largely because an experimental system cannot operate with a truly efficiently organized memory. A practical implementation of an n-gram system would be one which could respond with an entire row, column, or sub-array of bits in

a single memory-fetch cycle. There are three ways in which this could be implemented for, say a trigram system. The first would require special hardware arrays which when given two coordinates, would respond by supplying the sub-array of data corresponding to all possible values of the third coordinate. The second is a content addressable memory which could be implemented to respond in such a manner; thus far however, the cost of content addressable memories has been prohibitive (Foster, 1976). The third implementation would require only state-of-the-art hardware; a conventional, coordinate addressed memory (Foster, 1970) with a word size greater than 27 bits would store the sub-arrays as elements accessible by the address (n-gram number, coordinate-1, coordinate-2). This latter method would automatically triple the memory requirement of the n-gram; but it does have the advantage of only requiring state-of-the-art hardware.

CHAPTER VI

AN APPLICATION TO POSTAL OCR SYSTEMS

6.1. INTRODUCTION

This chapter presents a discussion of the Post Office mail sorting problem. A postprocessor which uses binary n-grams is applied to the context which is present in an address. In the latter part of the chapter, several experiments are performed to demonstrate the utility of the postprocessor design. The results clearly demonstrate the dramatic improvements possible through the use of context in this problem domain.

6.2. THE POST OFFICE PROBLEM

The Post Office problem is described very simply as the problem of sorting pieces of mail, e.g., 1) examining the address of an envelope or package, and 2) deciding the bin or mailbag into which the mailpiece should be placed. The various levels of detail necessary to the solution of the problem will be described in this section. It will be shown that, although the Zip Code is much simpler to read than the address, it is extremely difficult to detect an error once it has been made and impossible to correct the error without additional context.

6.2.1. One View of the Sorting Problem

In this subsection we present a simple view of the sorting problem. It differs slightly from that used by the Postal Service which is

described in the next subsection. The solution presented in this chapter is easily adaptable to the Postal Service sorting techniques, but it is couched in different terms because of the lower dimensionality inherent in the solution. Later it is shown that the difference between problem definitions appears to be only a semantic difference.

Consider first the problem of sorting outgoing mail. There are over 30000 Post Offices in the United States and each is potentially the destination of any outgoing letter. Hessinger (1962,1968) provides some indication of the distribution of outgoing mail, "About 80% of the mail [from a single Post Office] goes to only 100 cities and perhaps 98% goes to 1000 cities." Though he was speaking of the mail from Washington, D.C., he implied that a similar distribution would be found at any Post Office. If a machine which processes outgoing mail can correctly route all of the mail going to the 1000 cities and reject most (three-quarters) of the remaining 2% as unidentifiable, its error rate would be less than 1/2%. It seems reasonable to extrapolate that more than 99.5% of such mail goes to 10000 cities; if a machine could handle this proportion of mail well, it would have very low error and reject rates.

The problem of sorting incoming mail is somewhat different. Mail coming into a state must be sorted to from 200 to 2000 different destination post offices. Still, most of the mail goes to a few of the cities. It would be reasonable for a computer to know all of the Post Offices names within a state. Mail coming into a Post Office must be sorted by mail route and Post Office Box. To sort mail by route in some cities, a machine would have to know all of the relevant street names as well as the parts of streets allotted to each route.

Although the same OCR system design might be used to sort the mail regardless of the particular goals of the sort, the useful contextual information is different in each case. In the first two sorts described, the Zip Code is significant; if it can be recognized correctly and reliably, mailpieces can be sorted correctly. However, if even a single error is made in reading a Zip Code, there is a high probability (38%) that the resulting error will not be detected; if it is detected, it cannot be corrected with only the information which is present in the other four digits. But there is a great deal of additional information on the mailpiece for detecting and correcting such errors. The state name defines certain sets of characters which can be used as the first two digits of the Zip Code; the city name provides nearly perfect information for correcting the remainder of the Zip Code. The city name only fails to provide sufficient information for correcting the lower three digits of the Zip Code when the city is a multi-coded city (i.e., when the city has more than one Zip Code), but even then it can limit the set of potential corrections to a reasonable set. If mail is being sorted by state, the presence of the string "New York City" tells us that the error in "10044" is in one of the last two digits. (There is no 10044 Zip Code.) The strings "New York City" and "10044" provide sufficient information that a machine or human sorting the mailpiece can confidently route the mail piece to New York [State] or even to New York City.

One might wonder why so much emphasis is placed upon the recognition of the Zip Code if the state and city names provide so much more information. The reason is that, in general, more errors can be expected to be made on the state and city names than on the Zip Code[2]. On the other hand, the greater amount of context present in these strings makes them more easily correctable and, of course, the overall amount of information which is available from all three strings can be used to corroborate all decisions; i.e., there is a great deal of redundancy present.

6.2.2. The Actual Implementation of the Letter Sorting Problem

The use of context to improve the readability of mail is only a small part of the entire engineering effort required to sort mail. In the next few pages we shall present a brief overview of the organization of a Mail Transport Unit (MTU) and some of the associated software required to sort mail.

2 There are several general reasons for this. The most obvious is that there are 10 digits and 26 alphabetic characters; that is, there are fewer potential confusions for a classifier to make when reading digits, hence the process is inherently more reliable. Another reason is that the alphabetic strings are longer; therefore, even if the character error rate were the same, the string error rate would be higher for city names than for Zip Codes. Finally, lower case characters are read more poorly than upper case characters (or numerals) because they are (1) smaller (fewer pixels per image) and (2) more likely to have been typed by excessively dirty or damaged keys.

6.2.2.1. Hardware

The first component is the Loader. Letters are placed into the Loader by an operator. The mailpieces are placed here by the handfuls since they are to be processed at a rate of 12 per second (43200 per hour). The mailpieces are in position with their stamped edges trailing.

The Edger must place the lower and leading edges of envelcpes into a standard position and feed them to the MTU. Here, the mailpieces are jogged and blown to form loosely-packed bundles.

Pushers advance individual mailpieces to the feeder area. Sensors detect the thickness of the advancing mailpiece and adjust the speed of the pushers accordingly. The Feeder then must accelerate the mailpiece to a speed of 180 inches per second with 15 inches between the leading edges of the envelopes.

The Separator and Dedoubler must assure that multiple feeds (envelopes stuck together) do not get processed as a single item. Doubled envelopes are rejected into a stacker for later re-input to the system by the operator. Numerous safeguards must be implemented so that a jam or other machine malfunction either halts the machine components or re-routes the mailpieces out of the MTU rather than permit mailpieces to enter the jammed area and compound difficulties.

Ideally, mailpieces arrive at the scanning stations at the rate of 12 per second (or 12/n per second if n stations are multiplexed for

greater throughput). The scanning subsystems must locate all of the print on the mail piece, determine which is the address block, find the print lines within the address block, find the character patterns within the print lines, and store the associated data. Video preprocessing, if any, must be performed on the pattern data. This may include such miscellaneous functions as determination of line pitch, thresholding of grey tones prior to quantization, determination of character height and location within each line, and character isolation and normalization.

Finally character recognition is performed. Each character is processed by both a digit processor and an alphabetic character processor. Each of the two processors makes its best guess at the identity of each character in the aledged address block. For example, the address 1367 No. Pleasant St., AMHERST, MA 01002 might look like:

```
     4***85* *4 01002   1367 ** *1****** 5*
     AMHERST MA 0100Z   I*b* No Pleasant St
```

(the bottom line of the envelope is scanned and output first) where the top line is the output of the digit processor, the lower line is the output of the alphabetic processor, and stars(*) are used to indicate rejections by the respective processors. Note that some characters may be recognized as valid characters by both processors. It is also possible for a character to be rejected by both processors. There are other codes output by the alphabetic processor to indicate rejections but give some idea of the actual character identity; for example, the code "#" is used to indicate that the processor could not identify the character but "thinks" there is reasonable chance that it could be either an "N" or a "W."

The classification data is transmitted to the Directory Processor for use in identifying bin codes. The Directory Processor will be discussed later. The bin code identifies a location for the letter sorting machine to deposit mailpieces. The bins correlate one-for-one with the destinations of mail from the sorting Post Office. Some might be individual mail routes; some, individual nearby cities; some, sectional centers; and some might be groups of distant states.

6.2.2.2. The Directory Processor

The directory subsystem must process the dual output of the character recognition processors and determine a bin code. This subsystem performs several functions. The quantity of context used to verify correct reading of an address depends upon the amount of detail desired in the sorting of the mail. If the Directory Processor determines that the mailpiece is incoming, it is capable of performing an extremely complex set of searches to route the mailpiece to the correct carrier route. The directory subsystem uses information such as street lists and carrier route information to sort the mail. When a letter is recognized as having a destination in a nearby city (in the same or a nearby state), the actual lists of city names are used to verify correct reading of the mailpiece. To sort mail to distant states, little cross

checking of information is needed to verify sufficiently the correct reading of the address; for example, if the MTU is in New York, who cares if the street name in some distant city cannot be read correctly.

Although the actual algorithms used to search directories are not known to this writer, street, city, and state names are generally considered to have been found (matched) by the subsystem if the name, as read, contains fewer than 20% mismatches from the correctly spelled directory name.

There are numerous glitches which must be accounted for by such a subsystem. Letters are often addressed with no more locator information than "City," "NYC," or "NY, NY." Often the bottom address line is "attn" or "attention." Zip Codes may be either concatenated to or distinct from the state name or "City," etc.

6.3. FURTHER DEVELOPMENTS IN THE METHODS OF BINARY n-GRAMS

In this section we presented a method for correlation of city, state, and Zzip Code data. A trivial correlation could be obtained by merely concatenating the city name, state name, and Zip Code and treating the string formed as an extended word. However, this would result in a few intolerable conditions: the average string would be more than fifteen characters long; and several strings would have the same meaning, for example, "Amherst, MA 01002," "Amherst, Mass. 01002," and "Amherst, Massachusetts 01002." There must be some more efficient way to represent these strings without necessarily differentiating among those with the same meaning.

There is a lot of correlative information in the typical "City, State ZIP" string. Representing the Zip Code digits as $d_1d_2d_3d_4d_5$, each combination of the digits d_1d_2 is, with a few exceptions, restricted to a particular state. Similarly, given the state name and d_1d_2, the digits $d_3d_4d_5$ usually refer to a particular city; i.e., given the information "city, MA 01002," it is immediately known that "city" is "Amherst."

For convenience, the substring d_1d_2 will be referred to as a zip prefix and the substring $d_3d_4d_5$ will be called the zip suffix. The combination of the state name and the zip prefix will be termed the state name-zip prefix or SNZP. The combination of the city name and zip suffix will be termed the city name-zip suffix of CNZS.

6.3.1. A Simple Extension

In this section the method of binary n-grams is extended to include the notion of a fixed semantic contextual relationship.

Define a generalized n-gram as an n-gram which is based upon two related strings or substrings. For example, let us say that the n-gram $D_{ij/k}$ correlates the i-th and j-th characters of a primary string to the k-th character of the related string. In the context of this discussion the primary string is the city (or Post Office) name and the related string is the Zip Code. Thus $D_{14/3}$ correlates the digit d_3 of the Zip Code to the first and fourth characters of the city name. All of the n-gram types defined in chapters III and IV except the non-positional

n-gram can be extended to include this notion.

Substitution errors are detected, located, and corrected as in chapters III and IV. However, for the sake of consistency, the Pos function should be redefined. Define $\text{Pos}(D_{ij/k}) = \{i,j,z_k\}$, where z_k is used only to indicate that the referenced position is the k-th position of the Zip Code. All substitution errors are processed as they were in the previous chapters, regardless of the particular strings which is or appears to be in error.

The notion may be further extended by allowing $D_{i/jk}$ to correlate the i-th character of the primary string to the j-th and k-th characters of the related string.

Insertion, deletion, merger, and split errors are detected, located, and correlated as in chapter IV if they occur in the city name. If such an error occurs in the Zip Code, it will be seen later that the problems become more complicated; for the present, we will just insist that the Zip Code have five characters so that neither an insertion nor a deletion is possible.

6.3.2. The State Name

Generalized n-grams, as defined in the preceding subsection, may also be defined to correlate the state name to the Zip Code, in particular, the first two digits of the Zip Code.

The state name and the two-digit zip prefix (SNZP) can be processed differently from the city name-zip suffix (CNZS) discussed in the preceding subsection because, in the recommended form, they both have only two characters. In this form the notion of having an insertion, deletion, merger, or split error is a useless concept since a double substitution error is just as easy or difficult to correct. Suppose the unit "MA 02" is interpreted as "* 02," it does not matter what the error is called; if we are to insist that the "*" represents two characters ("**"), they must be either "MA," "RI," or "CT," because only these states have zips with "02" as a zip prefix.

The state name-zip prefix is different in another respect. The construct "MA 01" has precisely the same meaning as "Mass. 01" and "Massachusetts 01." Thus, while all are different lexical forms, they all have the same semantic interpretation; they are all synonyms for the same semantic entity, "Western Massachusetts." A practical system must recognize all of the possible forms. Since the longer forms can all be contextually recognized at least as well as the two-character form, we shall not consider them other than to say that the system can process them better than the two-character abbreviation.

Let us define a region as a set of state name-zip prefix groups. A region might be the single semantic unit

{MA 01} or it might represent the entire state

{MA 01,MA 02} or all of New England

{MA 01,MA 02,RI 02,CT 02,NH 03, etc.} , etc.

6.3.3. Joining The Techniques

Techniques for correlating some of the address information have been presented. We have shown how binary n-grams may be used to correlate Post Office names to three digits of the Zip Code and the state names to two digits of the Zip Code. An algorithm will be provided for correlating these two sets of strings. As we stated earlier, the state name may exist in several different forms; a means is needed for representing the semantics of each of the different forms. The method proposed will present a unique identifier for each <u>set</u> of one or more SNZP combinations.

Let a set of regions be defined and assign a unique identifier, a number S*, to each region. An n-gram triple will relate the region identifier (formed from the SNZP) to the pair of elements forming the CNZS. Define a generalized n-gram which includes the region identifier as a binary array $D_{*i/j}$ whose elements $d_{*i/j}(S^*,C_i,C_j)$ are 1 if and only if the corresponding individual characters C_i and C_j co-occur in some CNZS string in the region denoted by S*. For example, suppose that "Western Massachusetts" (MA-01) is encoded as "region 19." Then the string "Amherst, Mass. 01002" is encoded as "19/Amherst/002." Thus, $d_{*1/3}(19,A,2)=1$ while $d_{*1/3}(19,A,i)=0$ for i=4,5,6,7, or 8 since these combinations do not occur in "Western Massachusetts."

Detection of errors is the same as that defined earlier. However, correction of errors is different for the region identifier; that is, if it is detected to be wrong, as a substitution, correction must be attempted in a different procedure from that used in the earlier problems.

In the next paragraphs we discuss a few strategies for correction.

Suppose the machine reader decides that it has read "Amberst, MA 01002." The local context of the string "Amberst" or of the extended string "Amberst/002" may point out the error as being definitely in position three of the city name. If this is so, the system will correct the error or reject the string because correlation is ambiguous. However, if the error is not so well localized, e.g., if it is determined to be in either the third position or in the region identifier, what course should be chosen?

Suppose that the machine reader has recognized the string "Amherst, MA 02002" (the correct Zip Code is 01002) and that the string "MA 02" is identified as "Region 18" whereas the correct region is "19." Hopefully the processor would discover the fault in the extended string "'18'002/Amherst" with sufficient certainty that it would know that the error is in the region identifier. However, if it is determined that the region identifier is in error, what is to be done. If the error were to be treated as any other substitution error, the CPP would simply use the set difference rule: "If it's not '18,' it's one of the others -- any one of the others." In this case, that is the inefficient way to do it. There is a hierarchy of alternatives. It is more likely

that "MA 02" is a garbled form of "MA 01" than "UT 84"; the difference in relative likelihoods should be taken into account.

6.3.4. The Pragmatics of Processing

To incorporate the foregoing hierarchy into a CPP, the hierarchical decision structure in Figure 6.1 was selected. The SNZP is scanned for apparent errors; if no errors are apparent, the corresponding region identifier is found and the resulting CNZS-region identifier extended name is transmitted to the CNZSRI processor for scanning. If no errors are apparent, the string is accepted as it was originally input. If errors are apparent, correction is first attempted within the CNZS part of the string. If correction of a small number of errors (1 or 2) is possible within the CNZS, the correction is made and the string is output from the CPP. If more than one correction of the same type of error, e.g., a double substitution error, is possible, the string is rejected as uncorrectable due to ambiguities in correction. If no correction of a small number of errors in the CNZS is possible, or if all attempts at correction indicate that the region identifier is the "position" in error, the region identifier is rejected and the string is returned to the SNZP processor for further scanning.

If at first the SNZP processor accepted the SNZP as correct, it must now determine the set of all possible SNZP's which are one substitution different from the original SNZP. The region identifiers for

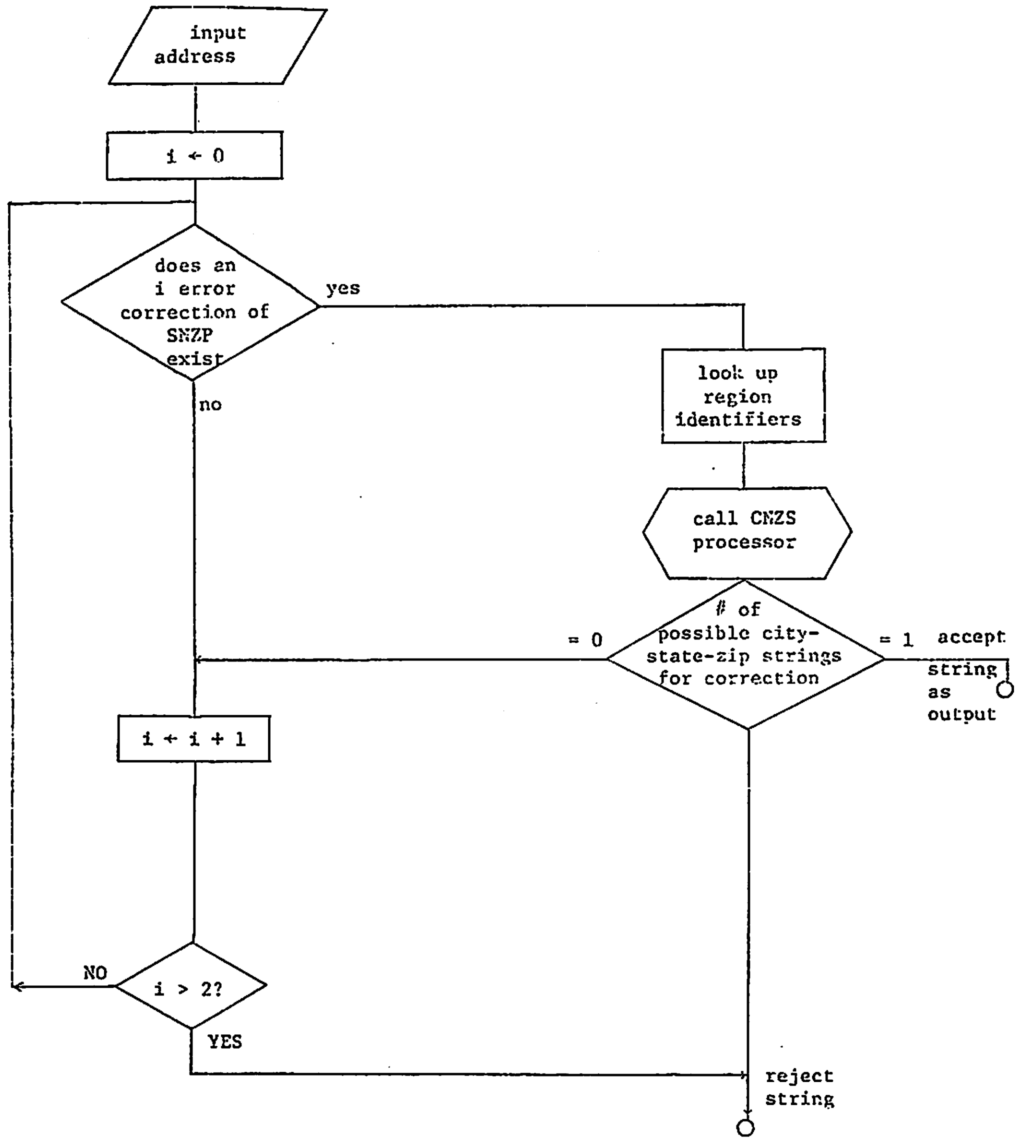
input address
i ← 0
does an i error correction of SNZP exist
yes
no
look up region identifiers
call CNZS processor
# of possible city-state-zip strings for correction
= 0
= 1
accept string as output
i ← i + 1
NO
i > 2?
YES
reject string

Figure 6.1 Decision structure flow diagram.

this set of SNZP's are then proposed as possible region identifiers for the particular CNZS. The CNZS and the set of region identifiers are then transmitted to the CNZSRI processor for correction of possible errors in the CNZS and selection of the correct region identifier.

Again, if no corrections can be made the string is either corrected or rejected, respectively, depending on whether one or more corrections are possible. If corrections cannot be made, the SNZP processor attempts correction of a double error in the original SNZP and outputs the appropriate set of region identifiers to the CNZSRI processor. If all this effort fails to find any corrections, the string is rejected.

6.4. <u>SOME EXPERIMENTS</u>

The system described in the preceding section has been aimed primarily at recognition of the SNZP part of an address. In fact, little effort has been made to resolve <u>ambiguities</u> in the CNZP part of an address. For example New York City is a multi-coded city (i.e., it has many [44] Zip Codes). "New York, NY 1001*" cannot be successfully resolved by the system proposed; proper recognition would be possible if the additional contextual information "E 47th St." were known. Therefore, given that the emphasis is on SNZP recognition, it must yet be shown that the additional algorithms are necessary to improve recognition of the SNZP as well as many CNZS strings.

The model of mail distribution used in these experiments is that mail to be sorted is distributed among the states as are the destinatio post offices; i.e., states with more post offices have more mail. The previous experiments in this paper and others have generally assumed that all words have equal probability of occurrence because it is easier to model the associated problem. In this chapter, the city-state-zip strings will be assumed to be uniformly distributed; this means that the multi-coded cities will be represented once for each Zip Code in the city. (Thus New York will occur 44 times in the data base.) Other, non-multi-coded Post Office names included in the experiments are represented once.

6.4.1. Region Identification

Several experiments were conducted to determine the correctability of the single and double substitution errors in the SNZP strings, both with and without the additional context of the CNZS. A few simplifications are made here. The first is that all state names are represented by their recommended two-letter abbreviation. Clearly, if other names or abbreviations are used, contextual recognition is even simpler since all names and all but one non-standard abbreviation are longer than two letters; Ohio has the only one letter abbreviation. Also, the region identifiers have been implemented in such a manner that a subsystem which uses synonyms would be an easy addition. The second assumption is that, for these experiments, identification of the last digits of the Zip Code is of secondary importance; identification of regions and city names is more important.

Some preliminary experiments were conducted to correct single and double substitution errors in the SNZP with no additional contextual information; i.e., the first two digits of the Zip Code are correlated to the state name but no CNZS information is used. Only 82% of single substitution errors and 45% of double substitution errors are unambiguously correctable. With the additional context of a correct CNZS these rates are raised to 99.1% and 92.0%, respectively; the corresponding error rates were .19% and 1.00%.

6.4.2. String Identification

A few experiments were performed to recognize typewritten characters (the CAL-U.S. Postal Service Alphanumeric Character Data Base [CAL, 1971]). The techniques (features, feature selection algorithm, Bayes' classifier) used closely paralleled those used in prior research (Fisher, et al., 1974). From this it was determined that the numeric characters could be recognized with error rates of 2 to 2-1/2% and alphabetic characters with error rates of 3-1/2 to 4%. Test data were then randomly generated for the following experiments using a digit substitution error rate of 3% and an alphabetic substitution error rate of 4%; these "rounded up" error rates were chosen so that "large" string error rates might be obtained from the simulated classifier.

6.4.2.1. A National Data Base

To simulate a sort of outgoing mail, the list of all first and second class post offices and their associated Zip Codes was used as a dictionary; random errors were generated at the above rates. The resulting distribution of errors is presented in the first column of table 6.1. Nearly 44% of the city-state-zip strings have errors in them and most of these strings with errors have only one error.

TABLE 6.1

RESULTS OF APPLYING CONTEXTUAL POSTPROCESSING TO A LARGE DATA BASE

Recognition Category	As output by the classifier		As output by the CPP		
	Correct	Error	Correct	Error	Reject
Region Identification	89.79%	10.21%	98.92%	0.04%	1.04%
Recognition of left 3 digits of Zip Code	91.05	8.95	96.65	0.72	2.62
Recognition of left 4 digits of Zip Code	88.16	11.84	94.45	1.46	4.07
Recognition of Zip Code	85.70	14.30	92.52	2.70	4.78
Recognition of city and region	61.51	38.49	97.20	0.11	2.68
Recognition of city, region, and Zip Code	56.25	43.75	91.12	2.78	6.10

Notes: 1) There were 17471 city-region-Zip strings in the data base; of these, 12652 were different cities.
2) The storage used for the CPP's data base was 4.3 million bits.
3) Only substitution errors were generated by the simulated classifier.

The set of n-grams used by the CPP consists of one non-positional quadgram, 98 forward-oriented n-grams and 98 backward-oriented n-grams. Twenty each of the forward- and backward-oriented n-grams correlated only city name positions; 63 correlated Zip suffix positions to city name positions; and 15 correlated city name positions to region identifiers. Although the total amount of storage required for all of these n-grams varies with the number of regions used, the same set of n-grams will be used for each of the experiments of this chapter.

The result of applying our contextual postprocessor to these data is presented in the second set of columns of table 6.1. These results are presented in several different ways because it is the ultimate design goal of the system which is important. That is, if the system is expected to unambiguously recognize all of the city-state-zip information, comparative statistics concerning expected behavior of the CPP will be found on the bottom line of table 6.1; if only regions must be unambiguously identified as well as possible the relevant data will be found on the top line of table 6.1. Thus, while it is interesting to know how the system may be expected to perform on the toughest formulation of the problem, it is also important to note how the system behaves when the problem is formulated more realistically.

Most (89.79%) of the strings output by the simulated classifier had the region identified correctly. Context corrected 89.4% of the 10.21%

strings in error to raise the recognition rate of 98.92%, leaving an error rate of 0.04%. Recognition of city names and regions is improved from 61.51% to 97.20; that is, the CPP unambiguously corrected 92.7% of the errors made by the simulated classifier on these strings. Zip Code recognition was not improved as well; many of the Zip Codes are difficult to correct because of insufficient contextual information from the multi-coded cities. The Zip error rate of 14.3% was reduced to 2.7%. Overall address recognition is raised from 56.25% to 91.12. Most of the strings which the CPP fails to recognize are those with ambiguities in the Zip Code. The difference between city-region recognition and city-region-Zip recognition is the set of strings which could not be corrected because of ambiguities in the Zip Code (68.47% of all such uncorrectable errors).

6.4.2.2. Some Statewide Data Bases

To simulate an incoming sort, some smaller experiments were conducted using data bases which consisted of all of the Post Office names of a single state. The three states chosen for use were: California, Pennsylvania, and New Hampshire. New Hampshire was selected because it is small. California and Pennsylvania were selected because they are both large. Pennsylvania has a much higher ratio of Post Office names to city-state-Zip strings (1816/2024 = .90) than does California (1154/1750 = .66); thus, since California has a higher proportion of

multi-coded cities the contextual recognition of zip codes might be expected to be different for these two states. The ratio of Post Office names to city-state-zip strings for New Hampshire is far higher (248/253 = .98); however, New Hampshire was selected because it is small and is expected to provide a different view of the CPP techniques. The distribution of errors generated by the simulated classifier and the results of contextual postprocessing these simulated data are presented in table 6.2.

In this problem it is more important that the entire Zip Code be read properly; the CPP does unambiguously recognize a greater percentage of Zip Codes. We see that the recognition rate for Zip Codes is far better for Pennsylvania than for California since Pennsylvania has fewer multi-coded cities. Most of the rejections in these cases are the strings which contain three or four substitution errors.

At this point the system is able to correctly recognize about 98% of the strings processed. In Letter Sorting Machines which use people to specify bin codes, the system goal is 95% correct sorts. The humans make, on the average, 2% error. (The mechanical parts of the sorting machine makes another 2% error.) Thus, the CPP performance is approximately equal to that of trained humans. If the CPP had the postal directory available to it, it would be able to do even better.

6.4.3. Other Error Types

In the preceding experiments of this chapter, only substitution

Table 6.2

CONTEXTUAL POST PROCESSING APPLIED TO SOME SIMULATED INCOMING SORT PROBLEMS

	Pennsylvania					California					New Hampshire				
	from classifier		from CPP			from classifier		from CPP			from classifier		from CPP		
Recognition of	Correct	Error	Correct	Error	Reject	Correct	Error	Correct	Error	Reject	Correct	Error	Correct	Error	Reject
Region	86.76%	13.24%	98.42%	.15%	1.43%	86.63%	13.37%	98.91%	.0%	1.09%	85.77%	14.23%	98.02%	.0%	1.98%
Left 3 digits of zip code	91.01	8.99	98.37	.20	1.43	91.54	8.46	98.80	.11	1.09	88.54	11.46	98.02	.0	1.98
Left 4 digits of zip code	88.64	11.36	98.12	.35	1.53	88.23	11.77	97.54	.80	1.66	87.75	12.25	98.02	.0	1.98
Zip code	85.42	14.58	98.07	.40	1.53	85.26	14.74	96.74	1.26	2.00	84.98	15.02	98.02	.0	1.98
City and region	60.77	39.23	98.17	.15	1.68	61.54	38.46	98.91	.0	1.09	65.22	34.78	98.02	.0	1.98
City, region, and zip code	54.55	45.45	97.82	.40	1.78	55.20	44.80	96.74	1.26	2.00	57.31	42.69	98.02	.0	1.98

Notes:	Pennsylvania	California	New Hampshire
Number of strings	2024	1750	253
Number of Post Office names	1816	1154	248
Bits of storage used by CPP (in millions)	2.30	2.36	2.23

errors were used. Substitution errors were used exclusively because of the lack of a model specifying how the other types of errors occur (and how they co-occur) in actual mail samples.

Given that only substitution errors occurred in the sample data, there were two design alternatives for the CPP. It could have been designed in a manner such that no attempt would be made to correct a probable error as either a deletion, insertion, split or merger. Else, it could be designed to attempt correction of suspected errors as if they might be any type of error. That is, the CPP may be designed either with or without a priori knowledge of the error types it is to encounter. In all of the preceding experiments reported, the CPP had no a priori knowledge of the fact that only substituion errors would be encountered. Some of the preceding experiments were repeated after altering the decision logic so that the CPP would not attempt to correct any strings processed as an insertion, deletion, etc. In the statewide-data base problems, this made no difference. In the case of the national data base, the correct recognition rate was off by about .01% (two strings were improperly corrected) but only in the city-name recognition categories since their regions and Zip Codes were properly recognized.

In chapter V it was noted that the CPP was able to correct strings with deletion errors least well. The experiments of this chapter used only substitution errors. In the absence of a model of the error generating processes, no experiments were performed in which all types of errors appeared. However, experiments were performed to determine whether or not the CPP could effectively process insertion and deletion errors. In the California data base, a randomly selected deletion error was generated in each city name; 1706 (97.49%) were unambiguously corrected. The remaining 44 (2.51%) were rejected as uncorrectable due to ambiguities in the correction of the city name; thus, in any recognition category which is not concerned with the precise recognition of the city name, accuracy would have been 100%. When randomly inserted errors were generated, only 5 of the 1750 insertion errors could not be corrected; again, however, all of the ambiguities were in the city names.

6.5. CONCLUSION

A discussion of the postal sorting problem was presented. It was shown that the problem may be seen from several points of view: there is a high level sorting of the mail in which mail is routed to large regions of the country; a lower level sort in which mail is routed to segments of these regions, (the segments are of town to large-part-of-a-city size); and a lowest level sort in which the mail is routed to individual mailboxes. It would be desirable to use machines to read and sort mail at least at the highest two levels because of the economies which might be realized by sorting such quantities of mail by machine. However, optical readers do make errors. Contextual post processing

techniques were proposed to find and correct these errors. The solutions proposed and examined in this chapter were shown to perform very well on the two highest levels. The reader simulated made errors on almost half of the strings processed. The CPP then corrected 80 to 95% of the strings in error (raising the correction rates to the 91 to 99% range and reducing error rates to .04 to 3%), depending upon the particular experiment design and goals. No effort was made to perform the lowest level sort since greater amounts of contextual information must be used to resolve ambiguities at this low level.

The proposed design for a CPP used several generalizations of the methods of binary n-grams. The strings correlated by n-grams were defined to include an additional component, a segment of the related Zip Code; both city names and state names were correlated to segments of Zip Codes. The city name-Zip suffix (CNZS) strings were also correlated to a region identifier.

CHAPTER VII
CONCLUSION

7.1. SUMMARY

It has often been shown that character recognition is a difficult problem. Humans and machines both have difficulty recognizing isolated characters; however, humans seldom read characters without regard to their context. Several attempts have been made to use context in the mechanical recognition of text. Many of them achieved significant improvements in their ability to recognize text.

In this thesis new methods of binary n-grams have been applied to the recognition of words. The new methods improve upon prior research with binary n-grams since they better facilitate the detection and correction of errors in words of different lengths. The additional types of n-grams developed in this thesis consider the letters, pairs of letters, and triples of letters in the context of their distance from either, or both, ends of words. They also facilitate the detection and correction of additional types of errors: that is, insertions, deletions, mergers, and splits as well as substitutions. The algorithms are relatively effective when processing these types of errors even though no prior information is available to the contextual postprocessor concerning whether the error is an insertion, deletion, merger, or split.

Chapter V presents the results of several experiments to determine the effectiveness of these new algorithms. For a dictionary of 10000

three- to sixteen-letter words, it was shown that the algorithms are very effective for substitutions, insertions, and splits; correction was over 80% for words with these types of errors. Words with a deletion or a merger error were corrected at least 50% of the time. Experiments were also performed to describe the effectiveness of the CPP algorithms for different sizes of dictionaries, different volumes of contextual information, and different mixes of n-gram types. The results cited above were for a dictionary of 10000 words; it was also shown that dramatically better results may be obtained when a smaller dictionary is used.

Chapter VI described a potential application of context in the recognition of text, namely used in a Post Office letter sorting machine. This is an ideal problem domain for application of context. The format of name-street address-city-state-Zip Code provides a conveniently fixed sentential syntax with a known (though large) set of elements for each component.

A specific solution to the use of context in the letter sorting problem is proposed. The solution adapts the methods and algorithms of binary n-grams to interrelate the city, state, and Zip Code components. The last three digits of the Zip Code, the Zip suffix, is correlated to city names in a manner which forms an extended word from the city name and the Zip suffix. Similarly, the state name and the Zip prefix (the first two digits of the Zip Code) are also linked to form another extended word. The earlier definitions of binary n-grams are modified to apply them to each of these extended words.

Sets of state name-Zip prefixes (SNZP's) are defined as regions. After detecting, locating, and correcting any errors in the SNZP, the region identifier which is identified by the SNZP is then correlated to the city name-Zip suffix (CNZS). Additional n-gram definitions are presented to correlate the region identifier to the CNZS. When an SNZP is found to be in error, there may be more than one region as alternative corrections; the algorithms developed resolve this type of ambiguity as well.

Experiments were performed to examine the effectiveness of the generalized algorithms; it was shown that the algorithms significantly improved the performance of the simulated pattern recognition system. With a national data base of over 12000 cities and 17000 city-state-Zip combinations, 43% of the strings output by the simulated classifier had errors in them; 80% of these errors were corrected, leaving a final, system error rate of 2.8% with 6.1% rejections. Most of the 6.1% which were rejected were ambiguities due to cities which contained more than one Zip Code; there was insufficient context to correct these Zip Codes. However, it might be noted that a letter sorting machine in New York is not concerned with correctly recognizing the precise zone of Los Angeles as long as it can send the mailpiece to California. With this in mind numerous other analyses of the CPP's effectiveness were presented. The

same simulated classifier committed errors on only 30% of the city-state-region strings which it processed; 93% of these errors were corrected, leaving a final error rate of 0.11% with 2.68% rejections.

The most crude sort desired by the Postal Service is by states (or regions). Ten percent of the strings output by the classifier had errors in the SNZP. Using all of the city-state-Zip information, whether correct or not, the CPP reduced the SNZP error rate to .0-%; 1.04% of the strings were rejected, mostly because they had too many errors.

Several experiments were also performed to determine the effectiveness of the algorithms on some state-wide data bases; the Post Office names and Zip Codes from Pennsylvania, California, and New Hampshire were used. In the Pennsylvania experiment, the classifier again output 45% of the strings (city-state-Zip) with errors in them; this error ratio was subsequently reduced to .4%.

7.2. SUGGESTIONS FOR FURTHER RESEARCH

7.2.1. Insertions vs. Deletions, Splits vs. Mergers

In section 5.3.1 it was seen that insertion errors are significantly easier to detect and correct than are deletion errors (95% of insertions are detected and 92% are corrected, versus only 76% and 57%, respectively, for deletions). Split errors are also easier to detect and correct than merger errors (97 and 81% versus 65 and 52%, respectively). These results suggest that a pattern recognition system should probably be geared to favor the production of insertion or split errors instead of deletions or mergers. Of course, some heuristics would have to be employed so that the system would not blindly classify all available noise as characters to produce words with many insertion errors. Also, any combination of errors is less desirable than one or no errors. Determination of a set of parameters which would lead to an optimum balance of errors is a topic for further research.

7.2.2. Further Postprocessing

The research of Hanson, Riseman, and Fisher (1975) used a decision mechanism to resolve ambiguities output by the CPP. Since the pattern recognition system they simulated made only substitution errors, the disambiguator only had to make some relatively easy decisions: letter C_i in position p_i versus letter C_j in position p_j (where p_i may equal p_j); letters C_{i_1} and C_{i_2} in positions p_{i_1} and p_{i_2} versus another pair of letters in an other pair of positions; etc.

We have defined a more comprehensive problem to deal with the additional complexities due to the presence of additional types of errors; the associated disambiguator must be able to process these additional types of errors. It must have a priori knowledge of the relative likelihoods of all insertions, deletions, splits, and mergers. It should also have a facility for merging split characters or splitting merged characters in order to classify the result. That is, if the CPP should claim that the characters which were classified as "VI" should be

either "W" or "E," the post-postprocessor should have a facility to determine the relative likelihoods $P(\text{"W"} \mid \text{"VI"})$ and $P(\text{"E"} \mid \text{"VI"})$ through a priori knowledge of such errors and/or through reclassification of the pattern. (Reclassification of the pattern with information about the prior misclassification forms a feedback loop to the classifier.)

7.2.3. Further Development of Contextual Algorithms

In chapter VI, it was shown that the recognition of state names could be improved by using the left two digits of the Zip Code. It was also shown that use of the city name and the remainder of the Zip Code could further improve a machine's ability to recognize the state name and the first two digits of the Zip Code, even if there are errors in the city name. However, little could be done to detect or correct errors in the rightmost digits of the Zip Code of a multi-coded city; it was shown that this incapacity made little difference if the mail was being sorted as outgoing mail to regions.

After reviewing some of the strings output by the CPP in the experiments of chapter VI, we would conjecture that Zip Code recognition for the national data base would be 96% (instead of 91) if the Zips of the multi-coded cities were not rejected because of ambiguities. If a dictionary of all of the necessary city names and their associated Zip Codes were used to resolve ambiguities and assist in correction, about 98.5% of the addresses would have been read correctly with about .2%

error and 1.3% rejections (neglecting Zip Code ambiguities). With additional postprocessing (as suggested in section 7.2.2) this could be further improved. If correction of additional error types such as triple substitutions were to be attempted, these error and reject rates would be further reduced.

Let us briefly describe a way to sort mail to the various Zip-Coded areas of a multi-coded city. Define each Zip-Coded area of a multi-coded city to be a region. Then form an extended word from the street address. Using the n-gram techniques defined in chapter VI, the region identifier (city zone) may be correlated to the extended word (street address) to detect and correct errors in the street address. This adaptation of the algorithms could be used both to resolve ambiguities in the low order digits of the Zip Code and to route mail within cities.

7.2.4 Other Application Domains

The algorithms presented in this thesis have been applied to the detection and correction of various types of errors in sequences of alphabetic characters. A similar but more difficult problem is the acoustical recognition of words in speech recognition. Although there may be many ways to formulate the problem, one can think of speech units as phonemes and sequences of phonemes as words. Mechanical processing of these sequences suffers from many similar problems such as ambiguities, substitution errors, insertions, deletions, mergers, and splits.

The binary n-gram techniques developed here may provide a reasonable solution to the problem of recognizing limited vocabulary speech.

Ehrich (1973) applied binary n-grams to the problem of dynamically segmenting handwritten text. The additional types of n-grams developed in this thesis may be adapted to the dynamic segmentation algorithms for use in speech and script recognition. Because these problems are so difficult, the improvement obtained through the use of contextual methods may be even more dramatic than the improvement obtained in the problems presented in this thesis.

BIBLIOGRAPHY

Abend. K. "Compound Decision Procedures for Unknown Distributions and for Dependent States of Nature," Pattern Recognition, ed. L. N. Kanal, Washington, D.C.: Thompson, (1968). 204-249.

Alberga, C. N. "String Similarity and Misspellings," CACM, 10. No. 5 (1967), 302-313.

Alt, F. L. "Digital Pattern Recognition by Moments," JACM, 9. No. 2, (1962), 240-258.

Alter, R. "Utilization of Contextual Constraints in Automatic Speech Recognition," IEEE Transactions on Audio and Electroacoustics, AU-16, No. 1 (1968), 6-11.

Blair, C. R. "A Program for Correcting Spelling Errors," Information and Control, 3 (1960), 60-67.

Bledsoe, W. W. and I. Browning, "Pattern Recognition and Reading by Machine," 1959 Proceedings of the Eastern Joint Computer Conference, 16, 225-232.

CAL. "An Alphanumeric Character Pattern Data Base (A Description for Potential Users and other Interested Parties," CAL Report No. XM-2586-X-4, Buffalo, N.Y.: Cornell Aeronautical Laboratory, Inc., (14 June 1968), Revised 8 Sept. 1971.

Carlson, G. "Techniques for Replacing Characters That Are Garbled on input," Proceedings of the Spring Joint Computer Conference, 30, (1966), 189-192.

Casey, R. G. and G. Nagy. "An Autonomous Reading Machine," IEEE Transactions on Computers, C-17, No. 5 (1968), 492-503.

Caskey, D. L. and C. L. Coates, "Machine Recognition of Handprinted Characters," Technical Report No. 126, Information Systems Laboratory, University of Texas at Austin, 1972.

Chandrasekaran, B. and A. K. Jain, "Independence, Measurement Complexity, and Classification Performance," IEEE Transactions on Systems, Man, and Cybernetics, SMC-5, No. 2, (1975), 240-244.

Chu, J. T. "Error Bounds for a Contextual Recognition Procedure," IEEE Transactions on Computers , C-20, No. 10 (1971), 1203-1207.

Chu, J. T. "Author's Reply [to Toussaint, 1972]," IEEE Transactions on Computers, C-21, No. 9 (1972), 1027-1028.

Conway, R. W. and W. L. Maxwell, "CORC -- The Cornell Computing Language," CACM, 6, No. 6 (June 1963), 317-321.

Conway, R. W., W. L. Maxwell, and R. J. Walker, An Instruction Manual for CORC--The Cornell Computing Language, Ithaca, N. Y.: Cornell University Print Shop, Sept. 1963.

Cornew, R. W. "A Statistical Method of Spelling Correction," Information and Control, 12, (1968), 79-93.

Cox, C., B. Blesser, and M. Eden, "The Application of Type Font Analysis to Automatic Character Recognition," Proceedings of the Second International Joint Conference on Pattern Recognition, Copenhagen, Denmark, August 13-15. (1974), 226-232.

Damerau, F. J. "A Technique for Computer Detection and Correction of Spelling Errors," CACM, 7, No. 3 (1964), 171-176.

Davidson, L. "Retrieval of Misspelled Names in an Airlines Passenger Record System," CACM, 5, No. 3 (1962), 169-171.

Devoe, D. B. and D. N. Graham, "Evaluation of Hand Printed Character Recognition Techniques," Rome Air Development Center, Griffiss Air Force Base, New York, RADC-TR-68-103, (1968).

Donaldson, R. W., and G. T. Toussaint, "Use of Contextual Constraints in Recognition of Contour Traced Hand-Printed Characters," IEEE Transactions on Computers, TC-19, No. 11 (1970), 1096-1099.

Duda, R. O. and P. E. Hart, "Experiments in the Recognition of Hand-Printed text: Part II--Context Analysis," Proceedings of the 1968 Fall Joint Computer Conference, 34, (1968), 1139-1149.

Edwards, A. W. and R. L. Chambers, "Can A priori Probabilities Help in Character Recognition?" JACM, 11, No. 4 (1964), 465-470.

Ehrich, R. W. "A Contextual Post Processor for Cursive Script Recognition--Summary," Proceedings of the First International Joint Conference on Pattern Recognition, Washington, D. C. (1973), 169-171.

Ehrich, R. W. and K. J. Koehler, "Experiments in the Contextual Recognition of Cursive Script," IEEE Transactions on Computers, C-24, No. 2 (1975), 182-194.

Ehrich, R. W. and E. M. Riseman, "Contextual Error Detection," COINS Technical Report, 70C-4, University of Massachusetts, Amherst, (1971).

Even, S. Algorithmic Combinatorics, New York: Macmillan, (1973).

Fisher, E. G. Master's Project, University of Massachusetts, Amherst, (1974).

Fisher, E. G., A. R. Hanson, and E. M. Riseman, "Feature Selection Using Non-Redundant Thresholded Measures," COINS Technical Report 74C-9, University of Massachusetts, Amherst, (Dec. 1974).

Forney, G. D. "The Viterbi Algorithm," Proceedings of the IEEE, 61, No. 3 (1973), 268-278.

Foster, C. C. Computer Architecture, New York: Van Nostrad Reinhold, (1970).

Foster, C. C. Content Addressible Parallel Processors, to appear, (1976).

Frishkopf, L. S. and L. D. Harmon, "Machine Reading of Cursive Script," Information Theory, ed. C. Cherry, London: Butterworth, (1961), 300-316.

Fu, K. S. Syntactic Methods in Pattern Recognition, Academic Press, New York, (1974).

Giangardella, J. J., J. F. Hudson, and R. S. Roper, "Spelling Correction Using a Digital Computer," IEEE Transactions on Engineering Writing and Speech, EWS-10, No. 2 (1967), 57-62.

Glantz, H. T. "On the Recognition of Information with a Digital Computer," JACM, 4, No. (1957), 178-188.

Gold, B. "Machine Recognition of Hand Sent Morse Code," IRE Transactions on Information Theory, IT-5, No. 1 (1959), 17-24.

Hanson, A. R. and E. M. Riseman, "System Design of an Integrated Pattern Recognition System or How to Get the Best Milage Out of Your Used Pattern Classifier," COINS Technical Report 73C-5, University of Massachusetts, Amherst, (June 1973).

Hanson, A. R., E. M. Riseman, and E. G. Fisher, "Context in Word Recognition," Pattern Recognition, to appear.

Harmon, L. D. "Automatic Reading of Cursive Script," Optical Character Recognition, eds. Fischer, Pollock, Radack, and Stevens, Washington, D.C.: Spartan Books, (1962), 151-152.

Harmon, L. D. "Automatic Recognition of Print and Script," Proceedings of the IEEE, 60, No. 10 (1972), 1165-1176.

Harmon, L. D., and E. J. Sitar, "Method and Apparatus for Correcting Errors in Mutilated Text," U. S. Patent 3 188 609, issued June 8, 1965.

Heinselman, R. C. "Computerized Detection and Correction of Spelling Errors in FORTRAN Programs," M. S. Thesis, Department of Information Science, University of Minnesota, Minneapolis, (1972).

Hennis, R. B. "The IBM 1975 Optical Page Reader, Part I--System Design," IBM Journal of Research and Development, 12, (1968), 346-353.

IBM, "Electro-Optical Recognition Techniques," RE 140-68, T. J. Watson Research Center, IBM, (July 31, 1969).

Jackson, M. "Mnemonics," Datamation, 13, No. 4 (1967), 26-28.

Kanal, L. "Patterns in Pattern Recognition, 1968-1974," IEEE Transactions on Information Theory, IT-20, No. 6 (1974), 697-722.

Kuvera, H. and W. N. Francis, Computational Analysis of Present-Day American English, Providence, R. I.: Brown University Press (1967).

Lin, W. C. and T. L. Scully, "Computer Identification of Handprinted Characters with a High Recognition Rate," IEEE Transactions on Systems, Man and Cybernetics, SMC-4, No. 6 (1974), 497-504.

McClusky, E. J. Introduction to the Theory of Switching Circuits, New York: McGraw-Hill (1965).

Mermelstein, P. and M. Eden, "A System for Automatic Recognition of Handwritten Words," Proceedings of the Fall Joint Computer Conference, 26, (1964), 333- 42.

McElwain C. K. and M. B. Evens, "The Degarbler--A Program for Correcting Machine-Read Morse Code," Information and Control, 5, (1962), 368-364.

Miller, G. A. and E. A. Friedman, "The Reconstruction of Mutilated English Texts," Information and Control, 1, No. 1 (1957), 38-55.

Morgan, H. L. "Spelling Correction in Systems Programs," CACM, 13, No. 2 (1970), 90-94.

Mucciardi, A. N. and E. E. Gose, "A Comparison of Seven Techniques for Choosing Subsets of Pattern Recognition Properties," IEEE Transactions on Computers, C-20, No. 9 (1971), 1023-1031.

Munson, J. M. "Experiments in the Recognition of Hand-Printed Text: Part I--Character Recognition," Proceedings of the 1968 Fall Joint Computer Conference 34, (1968), 1125-1138.

Nagy, G. "State of the Art in Pattern Recognition," Proceeding of the IEEE, 56, No. 5 (1968), 836-862.

Neisser, U. and P. Weene, "A Note on Human Recognition of Hand-Printed Characters," Information and Control, 3, No. 2 (1960), 191-196.

Neuhoff, D. L. "The Viterbi Algorithm As an Aid in Text Recognition," IEEE Transactions on Information Theory, IT-21, No. 2 (1975), 222-226.

Okuda, T., E. Tanaka, and T. Kasai, "A Method for the Correction of Garbled Words Based on the Levenshtein Metric," IEEE Transactions on Computers, C-25, No. 2 (1975), 172-178

Pratt, F. Secret and Urgent, Garden City, N.J.: Blue Ribbon Books, (1942).

Price, W. L. "Palantype Transcription by Computer--A Final Report," National Physical Laboratory, Com. Sci. 45, (1971).

Raviv, J. "Decision Making in Markov Chains Applied to the Problem of Pattern Recognition," IEEE Transactions on Information Theory, IT-3, No. 4 (1967), 536-551.

Riseman, E. M. and R. W. Ehrich, "Contextual Word Recognition Using Binary Digrams," IEEE Transactions on Computers, C-20, No. 4 (1971), 397-403.

Riseman, E. M. and A. R. Hanson, "A Contextual Postprocessing System for Error Detection and Correction in Character Recognition," COINS Technical Report, 72C-1, University of Massachusetts, Amherst (1972).

Riseman, E. M. and A. R. Hanson, "A Contextual Postprocessing System for Error Correction Using Binary n-Grams," IEEE Transactions on Computers, C-23, No. 5, (1974), 480-493.

Roberts, A. H. A Statistical Linguistic Analysis of American English, The Hague, Netherlands: Mouton and Co., (1965).

Sayre, K. M. "Machine Recognition of Handwritten Words: a Project Report," Pattern Recognition, 5, No. 3 (1973), 213-228.

Shannon, C. E. "Prediction and Entropy of Printed English," Bell System Technical Journal, 30, No. 1 91951), 50-64.

Shimura, M. "Recognizing Machines with Parametric and Non-Parametric Learning Methods Using Contextual Information." Pattern Recognition, 5, No. 2 (1973), 149-168.

Shillman, R. J. "Character recognition Based on Phenomenological Attributes: Theory and Methods," Ph.D. Thesis, MIT, (1974).

Sitar, E. J. Memorandum for File, Bell Labs, Sept. 12, 1961.

Szanser, A. J. "Error Correcting Methods in Natural Language Processing," Information Processing 68, ed. A. J. H. Morrell, Amsterdam: North Holland Publishing Co., (1968), vol. 11, 1412-1416.

Szanser, A. J. "Automatic Error Correction in Natural Languages," Information Storage and Retrieval, 5, No. 4 (1970a), 169-174. also in Statistical Methods in Linguistics, 6, (1970a), 52-59.

Szanser, A. J. "Resolution of Ambiguities by Contextual Word Repetition," Review of Applied Linguistics, 7 (1970b), 49-56.

Szanser, A. J. "Automatic Error Correction in Natural Texts--Part 1," National Physical Laboratory, Com. Sci. 46, (1971a).

Szanser, A. J. "Automatic Error Correction in Natural Texts--Part 2," National Physical Laboratory, Com. Sci. 53, (1971b).

Tanaka, E. and T. Kasai, "A Correcting Method of Garbled Languages Using Ordered Key Letters," Transactions of the Institute of Electronics and Communication Engineering o[illegible] pan, 55-D, No. 6 (1972), 363-370.

Tanaka, E., T. Kasai, and M. Fujino, [illegible] orrecting Method of Garbled Languages Using Language Infor[illegible] n," Transactions of the Institute of Electrical and Co[illegible] ication Engineering, 54-C, No. 4 (1971), 294-301.

Thomas, R. B. and M. Kassler, "Character Recognition in Context," Information and Control, 10, No. 1 (1967), 43-64.

Thorelli, L. E. "Automatic Correction of Errors in Text," BIT, 2 (1962), 45-62.

Thorndike, E. L. and I. Lorge, The Teacher's Word Book of 30000 Words, New York: Bureau of Publications, Teacher's College, Columbia University (1944).

Toussaint, G. T. "Comments on 'Error Bounds for a Contextual Recognition Procedure,'" IEEE Transactions on Computers, C-21, No. 9 (1972), 1027.

Toussaint, G. T. "Recent Progress in Statistical Methods Applied to Pattern Recognition," Proceedings of the Second International Conference on Pattern Recognition, Copenhagen, Denmark, August 13-15 (1974), 479-488.

Toussaint, G. T. and R. W. Donaldson, "Some Simple Contextual Decoding Algorithms Applied to Recognition of Handprinted Text," Proceedings of the Annual Canadian Computer Conference, Session '72 (1972), 422101-422116.

Troxel, D. Private Communication, (Aug. 1974).

USPS, "Zip-A-List Tape File Description," Postal Data Center, US Postal Service, (1973).

Vossler, C. M. and N. M. Branston, "The Use of Context for Correcting Garbled English Text," Proceedings of the ACM National Conference (1964), D2.4-1-D2.4-13.

Walker, R. J. An Instruction Manual for CUPL, The Cornell University Programming Language, Ithaca, N. Y.: Cornell University (1967).

A P P E N D I X A

DETECTION, LOCATION, AND CORRECTION OF DOUBLE DELETION ERRORS

This appendix outlines the techniques required for detecting, locating, and correcting double deletion errors in words. The intent of the presentation is to provide the reader with a grasp for the complexity of detecting, locating, and correcting multiple errors of the insertion-deletion-merger-split variety.

If $X=x_1x_2\ldots x_m$ is a word with deletion errors in positions r and q (r q), there exist letters y and y' and a dictionary word Y such that $Y=x_1\ldots x_{r-1}\ y\ x_r\ldots x_{q-1}\ y'\ x_q\ldots x_m$. Note that $Y=x_1\ldots x_{r-1}\ y\ y'\ x_r\ldots x_m$ is also possible (for r=q). Given a set of n grams $\mathfrak{S}$, the following statements are true:

a) for each D $\varepsilon\ \mathfrak{S}$ such that D(X)=0, $r \leq \max(\text{Pos}(D))$

b) for each B $\varepsilon\ \mathfrak{S}$ such that B(X)=0, $q > \min(\text{Pos}(B))$

c) for each C' $\varepsilon\ \mathfrak{S}$ such that C'(X)=0 $r \leq \max(\text{Pos}(C'))$
 and $r > \min(\text{Pos}(C'))$
 or $q \leq \max(\text{Pos}(C'))$
 and $q > \min(\text{Pos}(C'))$

d) for each E $\varepsilon\ \mathfrak{S}$ such that E(X)=0, $r \leq \max(\text{Posf}(E))$
 or $q > \min(\text{Posb}(E))$

The arguments presented in chapter IV for the single deletion error must be reworded here:

a) As in chapter IV, if $D_{ijk}(X)=0$, an error must precede k; since $r < q$, it must be that $r \leq k$. Unfortunately, nothing is implied concerning position q; i.e., either of the conditions $q \leq k$ or $q > k$ could be true. Thus, $r \in S = \bigcap_{\substack{D \in \mathfrak{D} \\ D(X)=0}} [1,\max(Pos(D))]$.

b) Similarly, if $B_{ijk}(X)=0$, $q > m+1-k$ and no information is provided concerning position r. Thus,

$q \in S = \bigcap_{\substack{B \in \mathfrak{B} \\ B(X)=0}} [\min(Pos(D)),m+1]$.

c) At least one deletion must be included among the set of positions indicated by any rejecting non-positional n-gram; however, this tells us little about the particular locations if there are several such rejecting n-grams. Suppose, that $C'_6(X)=0$ and $C'_7(X)=0$ and that C' is a non-positional trigram: if q (or r) is in position 8 nothing is known about r (respectively, q); if, however, r is 7, then q must be 8 or 9. For position location we shall resort to the tabular method used in chapter III to locate errors in words with two substitution errors.

d) At least one of the deletions must be among the two sets of positions indicated by an ends-oriented n-gram; however, just as the ends-oriented n-grams presented a few problems in localizing single deletion errors, they also are difficult to use for double deletion errors. The positions q and r will be determined in a manner similar to that used to find pairs of positions which might contain double substitution errors in section 3.3.3.

Suppose the following n-grams have rejected $X=x_1x_2...x_6$: D_{16}, D_{37}, B_{23}, B_{14}, C'_2, C'_4, C'_5, (C is a non-positional trigram), E_{14} and E_{23}. The corresponding chart is presented in table A.1.

TABLE A.1

EXAMPLE OF FIXING THE POSITION OF

DOUBLE DELETION ERRORS VIA n-GRAMS

	1	2	3	4	5	6	7
D_{16} and D_{37}	*	*	*	*	*	*	
D_{23} and B_{14}					*	*	*
C'_2			*	*			
C'_4					*	*	
C'_5						*	*
E_{14}	*			*	*	*	*
E_{23}	*	*			*	*	*

Possibilities for two deletion errors $\{3,6\}$ and $\{4,6\}$.

Note that it is possible to represent the forward-oriented and backward-oriented n-gram information each on single lines since, for example, the forward-oriented n-gram with the least greatest position (D_{16}) implies all of the positions which are indicated by any of the others; similarly, B_{23} is an implicant of B_{14}. Each of the rejecting non-positional n-grams and ends-oriented n-grams must be placed on lines of their own. The only pairs of positions which could contain deletion errors are $\{3,6\}$ and $\{4,6\}$.

Correction is attempted for each of these pairs of positions by "stretching" out the word X and attempting to find letters which will fit into the resulting gaps. (In the above example, the gaps will be in positions 3 and 7 or 4 and 7, respectively, for the sets $\{3,6\}$ and $\{4,6\}$).

APPENDIX B

ON THE LENGTH OF WORDS IN THE DICTIONARY

The intent of this appendix is to provide a comparison of the lengths of the words in the word lists of chapter V to the lengths of words in various lists of English words. The distributions of the lengths of the words in the four dictionaries used in chapter V are presented in table B.1. Since the research in chapter V assumes all strings to be equally likely, the mean length of the words in each of these dictionaries is 8.5 letters.

The length of the avarage word in a list of "most common words" depends heavily upon the size of the word list. Thorndike and Lorge (1944) published several word lists; one, a list of the thousand most common words of "standard English reading matter," had an average word length of 5.01 characters. Roberts (1965) used a list of 10065 words whose average length was 7.05 characters. Kučera and Francis (1967) published a corpus of 50406 different lexical units (most of which were words) with an average length of 8.13 letters.

In text, however, many words occur more often than others and the length of the average word is around 4.5 characters. Pratt (1942) cites the length of the average English word as 4.5 letters. Kuvera and Francis (1967) found an average word length of 4.74 characters in their total corpus of 1,014,232 lexical units (of which there were only 50406 different lexical units).

TABLE B.1

DISTRIBUTION OF WORDS BY LENGTH IN

THE FOUR SUBDICTIONARIES

word length	number of words in dictionary			
	1000	5000	10000	19196
3	20	27	55	107
4	34	176	352	676
5	76	397	795	1526
6	136	707	1415	2716
7	141	742	1479	2835
8	133	697	1393	2675
9	126	659	1317	2528
10	109	566	1131	2171
11	78	405	810	1554
12	54	283	567	1088
13	33	173	347	666
14	20	88	177	340
15	20	58	117	226
16	20	22	45	88

A P P E N D I X C

ABOUT THE IMPLEMENTATION

C.1 INTRODUCTION

In this appendix we describe a few of the methodologies used in the programs which simulated the proposed contextual postprocessors. An important consideration of the CPP is the storage management strategy employed. The n-grams used in chapter VI of this thesis differed from earlier n-grams in that they were of several different sizes; a variable size array structure was used to implement this feature. The descriptions of forward-oriented, backward-oriented, and ends-oriented n-grams were fairly complicated in their used of arbitrary subscripts. The implementation of this important feature is described in section C.3. An efficient search structure was needed to facilitate the finding of the set of all n-grams which concerned a particular position; the algorithm employed is described in section C.4. In the final section of this appendix we describe the algorithm used to implement the bit vectors which were continually "anded" with rows and columns of bit vectors from the n-grams.

C.2 The Variable Sized Bit Arrays

The n-grams which were used in chapter VI were of many different sizes. These arrays were implemented by building arrays of dope vectors.

Each n-gram's dope vector contains six elements: a pointer to the logical beginning of the n-gram array; the first two dimensions of the n-gram; and the position indices for which the n-gram (trigram) is to be used. Thus, if the n-gram element G(i,j,k) is to be referenced and G's dope vector contains the elements (A,B,C,D,E,F), the n-gram element (bit) G(i,j,k) is found by computing

$$(k*C+j)*B + i + A \quad .$$

The same formulation is used regardless of the ranges of i, j, and k; that is, if the first index of G refers to digits, $1 \le i \le 11$ (11 represents a reject character), etc.

The alert reader might note that the usual (FORTRAN) formulation of the above is

$$((k-1)*C+j-1)*B + (i-1) + A'$$

where A' is the physical origin of the n-gram G. However, if we let A = A' -B*C -B -1, the logical origin of the array is obtained and millions of subtractions are avoided. (On the Cyber 70/74 2.5 million subtractions only require one second so that the difference is moot anyway.)

C.3 String Representation

A 34 position vector was used to correlate CNZS with a region identifier. The Zip suffix was placed in positions 1, 2, and 3; the region identifier was placed in position 34; the city name started in position 7; and blanks were filled into the remaining locations. In this form the string was processed by both forward-oriented n-grams and non-positional n-grams. In this manner the general, forward-oriented n-gram $D_{25/3}$, which is to correlate positions 2 and 5 of the city name to position 3 of the Zip suffix, is actually represented as $D_{3,8,11}$ in the terminology of chapters III and IV. The reason for padding at least three blanks around the city name is that the non-positional quadgram can be generally applied to the city name without making exceptions for the ends of the word.

To implement backward-oriented n-grams without an incredible amount of bookkeeping to subtract indices from the length of words (plus one) and to keep the code as similar as possible, the strings implemented had the same format as those above except that the city name is spelled backwards starting in position 7. Thus, $B_{25/3}$, which is to correlate positions m-2 and m-5 of the m-letter city name and position 3 of the Zip suffix, can be more simply denoted as $B'_{3,8,11}$ within the program. (B' is used to denote an altering of subscript notations.)

To implement the ends-oriented n-grams, in the contextual post-processor of chapter VI (even though the experiments did not use the implementation), the Zip suffix was placed into the first three positions of a 14 element string; the first five characters of the city name were placed into positions 4 through 8; the last five characters were placed into positions 9 through 13; and the region identifier was placed into position 14. Note that a name less than 10 characters long will have some characters in two positions and that this anomaly must be considered when attempting to correct errors in these positions.

C.4 Lists of n-Gram Indices

To facilitate the execution of many of the algorithms, numerous list structures were built solely for answering the questions, "Which n-grams use position i?" or, "Which n-grams use positions i and j?" The trade-off here is the amount of time used to search the list of dope vectors for the desired indices versus the amount of storage required to contain the necessary pointers and counters. It is probably worth expending the storage since there were many cases in which no n-gram of some type referred to a particular position of some word. For example, if the greatest forward-oriented index implemented is 10 (or 16 by section C.3) any sixteen letter word has six positions which are irrelevant to the entire set of forward-oriented n-grams.

C.5 The Linked-List Bit Vector

In chapter V a few desirable methods of implementing storage for n-grams were discussed. Unfortunately, such a volume of storage was not available at the time of this writing. A more storage-conservative algorithm had to be designed. That is, we were unable to implement a system which could always respond in one memory cycle with a vector of 26 bits. We could have simulated the desired strategy but the cost of fetching one bit was so high that 26 repetitions of the fetch on the

Cyber 70/74 would still have required around 250 micro-seconds.

Alternately, a linked list was implemented. The list consisted of a counter of the number of non-zero elements in the vector, a pointer to the first non-zero element, and one pointer for each "non-zero" element in the list. Then, to "and" the vector with the desired 26-bit vector, one merely fetches the necessary elements of the 26-bit vector. (An element of the 26-bit vector is necessary if the corresponding element of the linked list is "non-zero.") Then, if the fetched bit is zero, the corresponding linked list element is deleted and the counter is decremented. Of course, the counter of non-zero elements is redundant since an empty list can always be represented by a "null" pointer; but the counter provides other bookkeeping routines with information they need.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER COINS TR 76-12	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* THE USE OF CONTEXT IN CHARACTER RECOGNITION		5. TYPE OF REPORT & PERIOD COVERED INTERIM
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Edward G. Fisher		8. CONTRACT OR GRANT NUMBER(s) N00014-67-A-0230-0007
9. PERFORMING ORGANIZATION NAME AND ADDRESS Computer & Information Science University of Massachusetts Amherst, MA 01002		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Office of Naval Research Arlington, VA 22217		12. REPORT DATE 7/76
		13. NUMBER OF PAGES 189
14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)*		15. SECURITY CLASS. *(of this report)* UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Distribution of this document is unlimited

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

character recognition
context
error detection
error correction
contextual postprocessor

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

In this dissertation new algorithms are developed for the application of context in character recognition. Improvements to binary n-gram algorithms are proposed which facilitate the use of different types of n-grams for a collection of words of varying lengths. Algorithms which utilize this extended data base for the detection, location, and correction of insertion, deletion, split, and merger errors are presented. A primary feature of the new algorithms is that the n-grams are anchored to one or both ends of the word.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

Experiments are performed which show that the algorithms are effective for all of the error types. For example, in a dictionary of 10000 words, without a priori knowledge of the types of errors being processed, the contextual postprocessor was able to correct 81% of single substitution errors, 57.7% of deletions, and 63.2% of double substitution errors. Numerous experiments were performed to examine the performance of the postprocessing algorithms as a function of dictionary size, as a function of word length, and as a function of the size of the contextual data base.

The application of these techniques to postal reading machines is explored in the latter part of this work. A brief exposition of the problem of mechanically sorting mail in the Post Office is provided. The algorithms are further developed to employ additional binary n-grams to correlate the redundant information in city-state-Zip Code addresses. They place a hierarchical emphasis upon the postprocessed data in order that the output may be viewed in the context of its relevance to the type of routing being performed.

Simulating a classification system which has a character error rate of 3 to 4%, errors were produced in 43% of the city-state-Zip addresses. The postprocessor was able to sort 97.5% of the erred addresses, yielding a gross sort to the correct region of 98.9% with an error rate of .04%. The CPP was able to sort 97.2% of the addresses to the correct city.