A COMPUTATIONAL METHOD FOR REDUCING WORDS TO THEIR GENERIC ROOTS UTILIZING A SUFFIX-EDITING ROUTINE

by

MARTIN LEON ZOLA

B. A., University of Notre Dame, 1967

A MASTER'S THESIS

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Statistics and Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1969

Approved by:

Major Professor

TABLE OF CONTENTS

Chapter Page

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

In spite of the limitations of a finite vocabulary, man is able to convey and to comprehend an infinite variety of messages. This unlimited capacity for verbal communication is due, in part, to a number of syntactical and verbal transformations which he employs to enlarge the range of meanings of each word in his vocabulary (6). One transformation of particular interest to this thesis is the utilization of suffixes as a linguistic device for extending the meanings of words.

DEFINITIONS

A suffix is any letter or string of letters which may be appended to the end of a word to produce a new word. The original word is referred to as the "parent" or "word root" and the new word (root plus suffix) is called the "whole word". More than one suffix may be afixed to a root at one time. Whole words with multiple suffixes may be reduced to a number of root forms. Suffix editing is the process of removing suffixes from words and trimming them down to their appropriate root form. A successful suffix edit is an edit which produces the smallest word root that in general has the same meaning as the whole word. This word form is referred to as the "generic root".

PROBLEM

Because of the overlap of semmantics between generic roots and full words, numerous text processing computer programs operate with word roots rather than whole words (12, 19, 21). Consequently, computational procedures for suffix editing play an important role in the efficient operation of this type of text processor. Unfortunately, suffix editing may also introduce a significant amount of error into the main process. Erroneous edits, such as failures to edit, inappropriate edits, and others, can seriously impede the performance of such systems--even if errors occur at a low rate. The problem then is to develop a fully self-contained routine which can make suffix edits that are comparable in accuracy to human performance.

Unlike humans, however, computer programs as yet do not "understand" the words on which they operate. A human, because he may refer readily to an extensive semmantic memory, is aware that HAT represents an apparel for the head and that HATING is an intensely negative emotional state. When asked to derive the root of HATING, a human, according to Associationistic Theory, solves the problem by implicitly tracing a chain of associations(7). The stimulus configuration HATING evokes a complex mental response to which other words are linked. By searching through these associated word forms, the person is able to select the appropriate root form, HATE. A computer program, on the other hand, can only respond to detailed instructions utilizing at best a very limited dictionary of associated word forms. Therefore,

rules for a computational method of suffix editing must be general enough so that they can identify when a string of characters is an editable suffix and, at the same time, have sufficient detail so that they can determine when two very similar situations call for different operations. A case in point is the pair HATS and HATING. The rules of suffix editing should be able to identify that S and ING are potential suffix strings. Furthermore, although both words appear to have the same base HAT, the rules should be able to produce HAT as the root of HATS and HATE as the root of HATING.

PURPOSE

The purpose of this report is to present a fully automated procedure which can be applied to most types of suffix and prefix editing problems. Since the procedure for editing suffixes can be extended to handle prefix editing as well, this thesis will be concerned solely with applications to suffix editing. The basis for this procedure is a comprehensive set of general and task-specific heuristics. The general heuristics represent a collection of guidelines for handling the global patterns appearing in a wide variety of suffix removal problems. As described in Chapter 2, there is a rich supply of problems to which a suffix-editing procedure can be applied. The general heuristics provide a consistent and highly effective approach to most of these problems. The task-specific heuristics relate to one type of problem only and must be developed by the user. A set of task-

specific heuristics have been developed for editing English language documents and will be described in detail in Chapter 3 of this thesis. The proposed procedure does not require dictionary look-up and can easily be modified to serve as an intermediate processor or subroutine.

An attempt has been made to derive empirical validation for the effectiveness of this computational procedure. A FORTRAN IV program, SUFCUT, has been developed incorporating the above heuristics. It is presented in Chapter 4. Several English language texts were run on the SUFCUT program. Median suffix length, percent savings, percent erroneous edits, percent edit ignores, and other quantitative measures were collected. The results of these experimental runs will be discussed in Chapter 5.

Chapter 2

APPLICATIONS OF SUFFIX EDITORS

This chapter presents a brief review of the ways in which suffix editing procedures are currently being utilized. Several new areas are identified as potential benefactors of this work. Finally, an attempt is made to bring some structure to the research in this area by proposing some improvements in the techniques of suffix editing.

COMPUTER CONTENT ANALYSIS

The General Inquirer

Perhaps, the most extensive application of suffix editors to date has been in the area of computer content analysis of verbal data. The forerunner of all editors was developed by Philip Stone and associates at Harvard for use with their General Inquirer Program System (22, 23). The General Inquirer is a content analysis program which tallies the frequency of occurence of various constructs and themes appearing in the data. During the processing of the experimental data, a suffix chopping scheme is applied to each full word producing a tentative word root. A dictionary of word roots is searched for an entry that matches the tentative root. Stone did not provide a detailed description of his editor, but he did imply that his technique was patterned after a standard suffix-editing procedure for English words (23:89). The standard procedure (which will edit no more than nine suffix-

es) is presented in flow chart form in Figure 1. No data are available describing the effectiveness of either technique.

Factor Analytic Methods

The studies of Harold Borko (3, 4, 5) in information retrieval have influenced psychologists Norman Haraway, Howard Iker, and John Starkweather to adopt a factor analytic approach to content analysis. In short, factor analysis is a complex statistical technique by which the covariance between numerous pairs of variables in the data are accounted for by means of a geometrically imposed structure of independent common factors (11). The factor analytic content analysis procedures use some type of suffix editing scheme in order to reduce the total number of different words in a document to a minimum number of root forms. Haraway and Iker (13: 5-6), without specifying any details about their procedure, have reported that their word editor STRIP generally reduces the number of different words in a document by almost fifty percent. Some of their documents have been as large as 25,000 words. Unfortunately, all edits made by STRIP had to be reviewed in order to keep the number of erroneous edits at a tolerable level.

In contrast with Haraway and Iker's STRIP program, Starkweather's approach has been non-computational in technique (19,20). He imposed a six-character limitation on the size of all words. Words greater than six characters in length were truncated beginning at the end of the sixth character. Starkweather has not

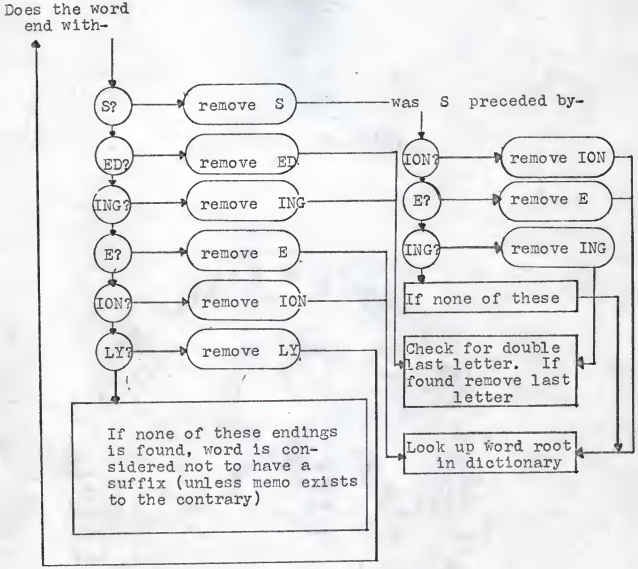


Figure 1

THE STANDARD SUFFIX-EDITING PROCEDURE USED BY STONE
AS A PATTERN FOR THE SUFFIX EDITOR
IN HIS GENERAL INQUIRER

reported the effectiveness of this approach. Apparently, he had assumed that if two words have the same first six consecutive characters, then they probably have the same word roots. One obvious advantage of this method is speed since it requires only one operation to be made upon each word. In addition, this technique might perform well on words with lengthly suffixes. But a significant portion of the words in the English language are less than or equal to six characters in length. Using Starkweather's technique, these short words will not be edited. Also, there may be some words whose roots are longer than six characters and these words will be lumped together although their root forms may very well be different.

LANGUAGE RESEARCH

FINDSIT

Suffix identification has made significant contributions in other areas of language research as well. Pylyshyn's FINDSIT has been used to determine the rate of occurrence of selected prefixes and suffixes during the course of psychotheraputic sessions (18). FINDSIT is a general purpose program for identifying parts of words, whole words, and groups of words in texts. FINDSIT may be used to reduce words to root form, although it has not been designed specifically for this purpose.

Syntactical Analyses of Sentences

Klein and Simmons (15) and Earl (8, 9, 10) have chosen

certain suffixes for analyzing the syntactical structures of English sentences. Suffixes, such as ION, ATION, OUS, LY, to mention a few, can be used to identify parts of speech of words. Klein and Simmons (15) have combined this type of suffix analysis with a sentence framing technique in order to code the grammar class of words in a sentence. The sentence framing technique was employed whenever a word did not have a suffix which clearly identified its part of speech. These authors reported that for two samples of scientific writing their approach had successfully identified the grammar class of over 90% of all words. Earl (10), using strictly a prefix-suffix analysis approach, reported that it is possible to determine with ninety-five percent accuracy the inclusive part of speech of an afixed word from a consideration of its prefixes, suffixes, and length. The inclusive part of speech was defined as "that string which contains all the parts of speech attributed to the word by the dictionary but which may also contain one or two more parts of speech" (10:53).

The computational methods used in these studies have been recently incorporated into several content analysis program systems (12, 21).

Scoring Projective Techniques

Working in the area of psychological testing, Donald Veldman and associates have developed a computerized version of the standard sentence completion inquiry (25, 26, 27). This test is a projective technique that was designed to be administered and scored entirely by computer. Their program has successfully

interacted with a large number of respondents via online typewriter console. The computer initiated the interaction on each item by typing out an incomplete sentence to which the respondent then replied with the first word that came to his mind. Each one-word response was chopped down by a suffix editor and an attempt was made to match it with a corresponding root form contained in a dictionary. Information necessary for scoring the response was obtained from the dictionary. The procedure was repeated until the entire battery of items was presented to each subject. Veldman found that the cost in using the computer for administering and scoring one respondent's protocol was "far less expensive" than if the procedure had been carried out manually by a qualified clinical pyschologist (27:9).

OTHER POTENTIAL APPLICATIONS

There are numerous additional applications for a suffix editor--many of which have not been fully explored. Research in the areas of language translation (8) and classification of organic chemistry nomenclature (24) have utilized some string identification procedures. But there are other areas such as automatic word hyphenating and document indexing which have as yet taken little or no advantage of the benefits of suffix editing.

SUMMARY

The applications of suffix editors reviewed in this chap-

ter are significant not only as a means to an end for some other supervisory procedure but are also valuable as ends in themselves. Standing alone, suffix editing represents one of the more challenging problems in the area of artificial intelligence. In effect, an attempt is being made to imitate a very specialized form of a most complex behavior phenomenon (language manipulation). What has been encountered here serves as a prelude to what might be faced in the production of higher order language processors.

Although several authors of language processing programs have attempted to develop computational methods of suffix editing, there is little information available in the literature which deals specifically with this problem. Most investigators have been concerned primarily with the performance of their main procedures, devoting little attention to the suffix editor which they employed as an important intermediate step. Little or no data has been published on the effectiveness of their techniques. In spite of this limited information, it appears that the performance of their routines can be characterized as being at a low level of sophistication. For instance, most of the current techniques have necessitated a reliance on a human monitor in order to reduce the number of erroneous edits. Furthermore, all of these techniques utilize a dictionary scanning procedure which in effect restricts the possibilities for successful edits to only those words whose roots can be found in the word root dictionary.

All techniques reviewed here have relied on a psuedo-memory or dictionary--no one has attempted to derive a set of

rules which are restricted solely to operations on the data without resorting to external aids such as dictionaries or human monitors. These procedures are highly specialized and are not capable of being liberally applied to many different types of problems.

The remainder of this thesis describes an attempt by this author to develop a set of heuristics for a computational procedure which will satisfy the above standards.

Chapter 3

HEURISTICS FOR SUFFIX EDITING

The heuristics to be presented in this chapter have been dichotomized into two groups. The larger group consists of general heuristics which are rules that, when taken together, dictate a style for solving most types of suffix-editing problems. However, by themselves they are not sufficiently detailed to edit any one type of data since it is unlikely that a single set of heuristics can be developed to the point that it could be applied to more than one heterogeneous type of natural language data. In order to adopt these heuristics to a specific problem, an interchangeable component must be included. This component is the set of task-specific heuristics. For each different problem, a new set of task-specific heuristics should be implemented.

DEFINITIONS

Task-specific heuristics define a set of suffixes to be edited and a corresponding set of follow-up operations which are enacted after a suffix is removed. The follow-up operations increase the flexibility and the power of the suffix editor. They enable the system to make multiple edits on the same word and also to add the vowels "E" or "Y" onto a root stem so that words like HATING can be reduced to HATE. Follow-up operations do not have to be performed after every suffix removal. The decision

as to when and where to apply them is at the discretion of the user.

The general heuristics do not change from problem to problem. Instead, they provide the context in which the task-specific heuristics function. They also prepare the data into a form which the task-specific heuristics can utilize. These objectives are accomplished by employing the following operations.

PROCESSING

The document to be processed is broken up into a list of variable length words. These words are sorted into alphabetical order and each is assigned an initial frequency of one. Starting with the first two elements at the top of the list, each pair of adjacent words are compared to determine: 1) if the words are the same, 2) if they share equivalent root forms, 3) neither of these. The problem of determining which words are identical is trivial. This can be accomplished by a serial comparison of all characters in both words. The real difficulty is in determining when two adjacent words have the same word root. By trimming away suffixes, both words may be reduced to the same root. However, the problem is to avoid under and over trimming words. Furthermore, a supplementary technique is necessary for editing those pairs of adjacent words which do not share the same generic root.

In order to solve these problems the following heuristics are employed when processing two adjacent words. The first

operation is to calculate the number of leading characters which both words have in common. If all characters are the same for both words, then the words are equivalent. When three or more (but not all) leading characters are shared in common by both words, the suffix removal process is activated. The reason for this cut-off criterion is that very few word roots, if any, are less than three characters in length. Thus, if two adjacent words fall short of this criterion, it is highly improbable that they will have the same root. In this case a special routine takes over. This routine is an abbreviated form of the regular suffix editor, containing only a few suffix strings which can be safely edited from one word. It edits only the first word of the pair. This special suffix editor is also applied to words which were unable to be reduced by the regular suffix editor.

Given the number of leading characters in common between two words, the suffix editor is capable of delimiting that part of both words which may be stripped off as suffixes. For instance, the words ACTIONS and ACTS do not match beginning with the fourth character. Consequently, potential suffix strings on both words extend from the fourth to the last character. In this example, IONS and S may be removed as suffixes. The next step is to search the list of editible suffixes for a matching suffix string. ACTIONS may be edited several times until the string of characters IONS is either completely stripped away or is reduced as much as possible.

Each time a suffix is removed from a word a new word form is produced. Two words are considered to have the same word root when any of their reduced word forms are equivalent. In the above example, the reduced word forms of ACTIONS were ACTION and ACT. Since ACTS is also reduced to ACT, then both words are assumed to have the same root form ACT.

One means of improving the performance of this procedure is to enact a follow-up operation after a suffix is deleted. Each suffix is assigned a particular follow-up beforehand. They may also be specified to conditions where no suffixes are edited. One obvious follow-up operation is employed whenever the suffix S is removed from a word. This operation designates making another edit on the remaining stem. Thus, ACTIONS is at first trimmed to ACTION and then, as a result of a follow-up command, reduced to ACT. Another type of follow-up procedure is to add a letter to a stem after removing a suffix. The follow-up after removing ING might be designed to generate two root forms, one with the stem and another with the stem plus "E" (or "Y"). In this manner, HATING can be reduced to two root forms HATE and HAT. If HATING is immediately preceded by HATE, then both words would be combined.

When a pair of words if found to be identical or to have equivalent root forms, then frequencies of occurrence of the second is added to the first and the second word is dropped from the list. The next pair to be compared should contain the first word of the last pair and the word immediately follow-

ing the one that was previously dropped. The procedure is continued in this manner until all pairs of adjacent words in the entire data base are processed.

Chapter 4

THE SUFCUT PROGRAM

SUFCUT is a FORTRAN IV program designed to edit suffixes from words in English language documents. The method of solution is based on the heuristics presented in the last chapter. Although at the present time it has been applied only to English language data, the program incorporates an interchangeable component which enables it to be extended to a broad range of suffix problems.

Structurally, SUFCUT consists of a main routine and three subroutines.[1] On a conceptual level, however, it can be viewed as consisting of four independent procedural steps. These procedural steps are outlined in Figure 2.

PROCEDURAL STEPS

Step One prepares the data for subsequent suffix processing. During this step, the data is read as a continuous stream and is formed into individual words that are stored sequentially into an array called WORD.

The data may be read from cards, tape, or disk device. If punched cards are used, the data is transcribed into columns one through seventy with an unrestricted format. Columns seventy-

---

[1] The source listing for the SUFCUT program is presented in Appendix 3.

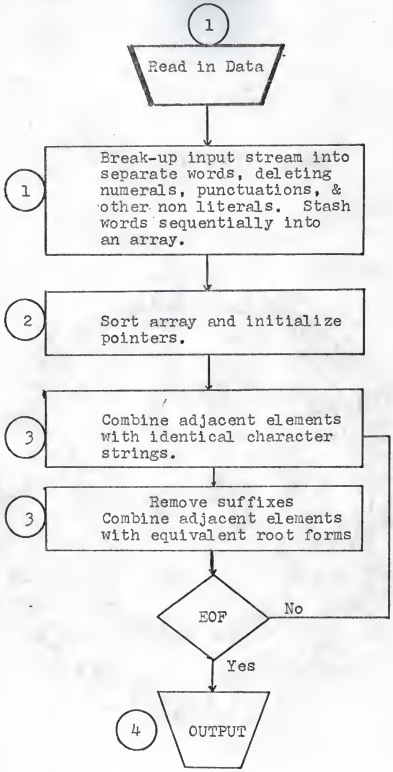


Figure 2

A CONCEPTUAL OVERVIEW OF THE SUFCUT PROGRAM
WITH OPERATIONS BLOCKED INTO PROCEDURAL STEPS

one through eighty are reserved for card sequence numbers and other types of identification codes. The only exception to this free form is that all words be separated from each other by at least one blank. Whenever the last letter of a word is positioned into column seventy, a blank must be inserted into the first column of the suceeding card. A word that is interrupted on column seventy without being completed must be continued starting with column one of the next card. Numerals, punctuation marks, and other non literal characters may be included in the data as they will be ignored by the program. The maximum size for each word is fifteen characters. The program automatically truncates any words exceeding this limit.

Step Two is a sorting procedure which is applied to the elements of the WORD array. The purpose of this procedure is to sort these elements into alphabetical order. A pointer array (PTR) is established so that given any element of the WORD array the next succeeding element can be located. This is accomplished by referring to that value of the PTR array whose index is the same as the original word. In order to designate the end of the WORD list, the PTR value of the last element in the WORD array is assigned a value of zero.

Step Three initiates the actual processing of the data. It is an iterative procedure that is performed on each pair of adjacent elements. During the execution of Step Three, if the words being compared are not equivalent, subroutine SUB1 is called. It in turns calls SUB2 and SUB3, SUB2 is the inter-

changeable component of this program. It contains the binary tree of editable suffixes and the corresponding set of follow-up operations. Each word presented to SUB2 is systemmatically stripped of suffixes. Each time a suffix is removed from a word a tentative root form is created. One word can be edited several times in succession.

SUB3 compares the tentative root forms of two words seeking to determine if any one form occurs for both words. When two words are found to have the same word root, the PTR array is adjusted so that the second word is dropped from the list.

Step Four prints out in alphabetical order all word roots and their corresponding frequencies of occurrence.

THE SOURCE LANGUAGE

Although FORTRAN is not a particularly suitable language for string processing, it does possess certain valuable attributes as the source language of SUFCUT. Perhaps, its greatest advantage is its "universality". No other programming language is as widely implemented as FORTRAN. Whereas other string-processing languages like COMIT, SNOBOL, and LISP (17:108) might simplify the programming task, their processors are currently rather limited in distribution. Another positive aspect of FORTRAN's "universality" is that it is well known throughout the scientific community. The user who wishes to alter SUFCUT will be able to make the modifications himself; or, if he prefers, he will not have much difficulty in finding someone else to do the job for

him.

MODIFICATIONS

In order to adapt SUFCUT to new types of problems, the program structure is altered in the following manner. The user begins by constructing an exhaustive list of prefixes, suffixes, or other character strings which he is interested in stripping from words in his data base. This collection of character strings is referred to as the strip list. In developing a procedure for suffix editing the following steps are taken. Since the procedure edits a word starting with its last character and working toward the first, it is necessary to reverse the order of characters of each string in the strip list. Suffixes having the same character for the first letter of the reversed string, are sorted together. Each group of suffixes with the same letter in the first position (formerly the last letter of the original suffix) is organized into a tree structure with the common letter as the root node. This process of building a tree structure for each suffix cluster is demonstrated in Figure 3. for suffixes ending with the letter N. The multibranching tree (Figure 3, Step C) is converted into a binary tree (Step D). Each node of the binary tree contains a value (SUFTST), a left pointer (LPTR), and a right pointer (RPTR). A node whose value is delta (Δ) represents a blank, indicating a termination of a suffix string. The circular nodes found in Figure 3 (Step D) represent exit values. If while traversing a tree an exit value is encountered rather than a pointer, the search process

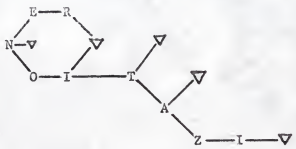
A. Suffixes ending with N
(sorted by size)

1. N
2. I-O-N
3. R-E-N
4. T-I-O-N
5. A-T-I-O-N
6. I-Z-A-T-I-O-N

B. Reverse order of characters

1. N
2. N-O-I
3. N-E-R
4. N-O-I-T
5. N-O-I-T-A
6. N-O-I-T-A-Z-I

C. Suffixes coalesced into a multibranching tree structure



D. Multibranching tree transformed into a binary tree with exit values

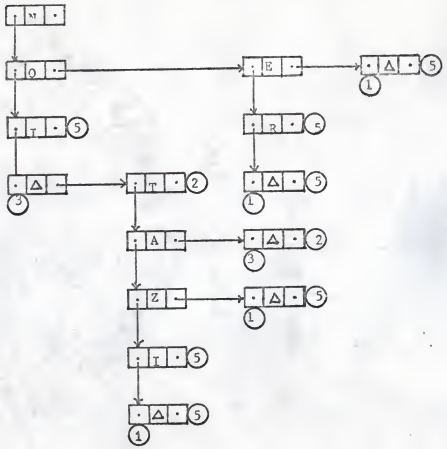


Figure 3

STEPS IN BUILDING A BINARY TREE STRUCTURE FOR N-ENDING SUFFIXES

is discontinued and the follow-up operation indicated by the exit value is enacted. Table 1 contains a complete listing of follow-up operations which may be performed in conjunction with the English language suffix editor of the SUFCUT program. After all suffixes have been built into the appropriate subtrees, the roots of all subtrees are linked together. The order in which the root nodes appear from left to right is usually adjusted according to how frequently subtrees will be searched. Subtrees containing the most frequently appearing suffixes will be situated to the left (before) subtrees with less frequent suffixes.

Suppose the string ION is to be trimmed from the word ACTION. The last letter in the suffix string is N. Starting with the list-head node of the entire SUFTRE, a comparison is made between N and the value of the list-head node. If the two values are identical, a branch is taken to the node indicated by LPTR (provided, of course, that LPTR does not indicate an exit value). If they fail to match, control branches to the node indicated by RPTR. Before preceeding with each branch, each pointer is checked to determine if it is an exit value. Suppose that the list-head node contains an S, then a branch would be taken to the node indicated by RPTR. The next node would be compared with N and so on until the node containing N is located. The node whose value is N is the root of the subtree containing all suffixes which end with N. If there were no suffixes in the strip list ending with N, the procedure would test all root nodes of all subtrees until the RPTR of the last

node is encountered. In such an event the last RPTR would contain the exit value five. Follow-up five indicates that the tentative suffix is not deletable (See Table 1). However, in the above example, N is a terminal letter in a suffix string; therefore, the N node is eventually located. A left branch is taken from the N node--after determining that it does not point to an exit value. At this time, the next character (O) backward in the suffix string (ION) is compared with the contents of the node indicated by LPTR of N (See Figure 3., Step D). They match so another left branch is taken. This time the value of the new node is compared with character preceeding the last character tested. This test is positive as I matches with I. The left branch is again taken. Now all leters allowed to be stripped as a suffix have been used up. When this situation occurs, one last left branch is made. If a delta node can be found without making another left branch, the exit value in LPTR of that delta node is taken. Otherwise, consecutive right branches are taken until either a delta or an exit value is encountered. In this example, the value of the node indicated by the LPTR of node I is a delta. Follow-up operation three is enacted (See Table 1) and the process is completed.

In the FORTRAN language, a binary tree structures like SUFTRE can be represented by three related arrays--ie. SUFTST, LPTR, and RPTR (16:360). Thus, one index which identifies a node on a tree serves as a subscript for all three arrays. As a result, the same index can retrieve information from any one

TABLE 1

ALL FOLLOW-UP COMMANDS INITIATED BY EXIT VALUES
IN THE SUFTRE OF THE SUFCUT PROGRAM

EXIT VALUE	O P E R A T I O N S
1*	Remove suffix string Put stem in STACK Terminate SUB2
2	Remove suffix string Put stem in STACK Make another pass through SUFTRE
3	Remove suffix string Put word stem+E into STACK Put word stem into STACK
4	Remove suffix string Put word stem+Y into STACK Put word stem into STACK Terminate SUB2
5	This is not a deletable suffix Terminate SUB2
6	Remove suffix string If last two letters of stem are equal, drop second letter and put new stem into STACK Otherwise, put first stem into STACK Terminate SUB2
7	Drop last letter of whole word Put remaining stem into STACK Make another pass through SUFTRE
8	Remove suffix string If last two letters of stem are equal, drop second letter and put new stem into STACK Otherwise, perform follow-up 3.

*Exit values are stored in the SUFTRE as negative integers in order to distinguish them from pointers (see page 22).

of these related arrays.

In preparing the SUFTRE for insertion into SUB2, literal characters are converted into the integer values which correspond to their alphabetical order. These integer values--and not literal characters--are stored in the SUFTST array. In addition, exit values are transformed into negative integers in order to render them distinguishable from other pointers. The entire SUFTRE structure (ie. the SUFTST, LPTR, & RPTR arrays) is inserted into SUB2 by means of a DATA definition statement.

The instructions carrying out follow-up operations should be contained solely in SUB2. Each time a branch is anticipated from one node to another; the pointer is tested beforehand for a negative value. A negative result indicates that the pointer is an exit value. The search of the SUFTRE is discontinued and the negative exit value is converted to a positive integer. This integer becomes an index for a Computed GO TO statement and the control branches to that part of the subroutine enacting the appropriate follow-up operation.

As mentioned earlier in this chapter, SUFCUT has been designed so that it can be extended to perform other types of natural language manipulations. In order to achieve this functional flexibility, a variety of programming techniques may be employed as needed.

More than one tree structure may be integrated into SUB2. For instance, along with the SUFTRE, a tree of prefix strings could also be included. This compound structure would then be

scanned in such a manner so that suffix and prefix editing could operate concurrently on the same data.

Another possible type of program modification would be to introduce a dictionary into the SUFCUT editing process. A dictionary might be used in any number of different ways. One possible method could be to store all word roots of interest in a dictionary. Each word in the text could then be tagged with the largest word root that can be found in the dictionary. The SUFCUT suffix editor could then be applied to each word in order to match it with its tentative root.

Chapter 5

EMPIRICAL VALIDATION

A series of brief investigations was conducted in order to obtain some data on the effectiveness of the SUFCUT program. In an preliminary study, SUFCUT was applied to a block of 300 words taken from an article on text processing by Berns (1:145). The results were compared with a set of word roots produced by hand editing the same data. Ten erroneous edits were identified, yielding a 3.3% margin of error. Most of the errors consisted of under edits (failing to strip away enough characters) and edit ignores (failing to edit when appropriate). None of these words was mistakenly combined with another word root. Furthermore, SUFCUT was successful in accurately reducing the original document to 220 word roots--a 26% savings.

Method

In order to cross validate some of these preliminary findings, two new samples of data were run on the SUFCUT program. This time, instead of the author editing the data, a committee of three independent graduate students was established for this purpose. Before doing any hand editing each person was presented a standard set of directions. The directions were augmented by several examples. Appendix 4 displays a copy of these directions.

The samples were drawn from two sources. The data for Sample A had been provided by Dr. David Danskin and Dr. Carroll

Kennedy of the Counseling Center, Kansas State University. Danskin and Kennedy have compiled a data bank consisting of 5,209 words transcribed from recordings of small group discussions. These words were stored in alphabetical order on a disk at the KSU Computing Center. The first 200 consecutive words from these records were selected for Sample A.

Sample B was obtained from the Davis Howe Word Count of Spoken English (14). In a recent publication, Howe (14) presented a glossary of 9,699 different words which were spoken during 250,000 words of recorded interviews. As in Sample A, only the first 200 words in this glossary were included in Sample B.

Results

Samples A and B were rated by human editors with the following results (refer to Tables 2, 3, and 4). The original corpus of Sample A had such a high incident of misspellings that 73 words had to be disqualified from further data analysis. Each of the remaining words appearing in this sample were assigned to a rater agreement (RA) category depending on how well the raters agreed in editing that word. With three raters, there can be three possible types of agreement: 1) No rater agreement--each rater disagrees with the other two, 2) a majority rater agreement--two out of three raters (2/3) make the same rating, and 3) unanimous agreement--all raters (3/3) make the same rating on the same word. The words from both samples are

Table 2

CLASSIFICATION OF WORDS IN SAMPLES A AND B
ACCORDING TO RATER AGREEMENT AND
TYPE OF PROCESSING

	SAMPLE A		SAMPLE B*	
RATER AGREEMENT:	Majority	Unanim.	Majority	Unanim.
Type of Processing				
1) No processing. Words already in root form.	7	48	10	98
2) Editing required. Words not in root form.	10	62	27	62
TOTALS:	17	110	37	160

*Three words in Sample B had no rater agreement.

Table 3

LENGTH OF SUFFIX STRINGS EDITED FROM WORDS IN SAMPLES A AND B

Suffix String Length	SAMPLE A		SAMPLE B	
	Majority Rated Words	Unanim. Rated Words	Majority Rated Words	Unanim. Rated Words
1	1	12	3	26
2	1	13	3	15
3	1	17	5	10
4	2	13	1	4
5	2	3	3	1
5	0	1	0	0
TOTALS:	7	59	15	56

Table 4

THE NUMBER OF WORDS IN SAMPLES A AND B
WHICH SUFCUT FAILED TO TRANSFORM
INTO THE SAME GENERIC ROOT AS
PRODUCED BY HUMAN EDITORS

	SAMPLE A		SAMPLE B	
RATER AGREEMENT:	Majority	Unanim.	Majority	Unanim.
Types of errors				
Over edits	0	0	2	4
under edits	0	2	8	2
ignores	3	1	2	0
TOTALS:	3	3	12	6

classified in Table 2 according to sample, rater agreement, and type of processing undertaken by the raters.

The raters' edits were taken as standards for evaluating the output of the SUFCUT program. If an analysis of SUFCUT's performance were based solely on those words which the raters processed in a unanimous fashion (RA=3/3), then the precentage of error occurring in both samples was approximately 2-4%. This figure seems to be in keeping with the 3% margin of error found in the preliminary study. Those words in both samples which the raters failed to achieve unanimous agreement were examined separately. If two of the raters agreed on a root for a word, then this word was put on a special list of majority (2/3) rated words. SUFCUT's percentage of error for editing words in the majority rated classification was between 17 and 32%.

One possible explaination for this increase in the error of SUFCUT's performance is that it might have been partly due to a statistical artifact. The raters themselves were at times uncertain about how to edit words. In the first sample they disagreed on 12% of all words. This figure jumped to 21% in Sample B. In effect, word roots generated by SUFCUT were being compared with a criterion that in itself was partly in error. The combination of these two types of error may have produced an attenuation of SUFCUT's true accuracy.

Words classified in the majority category appear to be longer and as a result may have been more difficult to edit. In Samples A and B, the words in the majority category had a median

character length of eight characters. Words in the unanimously rated category were shorter, having a median length of seven characters. A Chi-square test for differences between medians was significant at the .025 level. In addition, other data (Table 3) demonstrate that longer suffixes were being trimmed from words in the majority category. The median length of edited suffixes was three characters for majority rated words as opposed to two characters for unanimously rated words. The results of the Chi-square median test indicated that these differences were significant at the .10 level. These findings seem to indicate that the larger the word and the longer the suffix string the more chance there is for program error. These same variables may have also imposed a burden on the human rater as well. The task becomes more difficult as the word length and the suffix strings become larger because larger words can be reduced to a greater number of potential roots. Thus, the presence of additional potential roots requires that an editor possess some kind of method of determining which characters form the proper root. Unfortunately, it has not been possible to distinguish when the program, not the rater, failed to identify the proper root.

Savings scores, based on the reduction in the total number of different words due to suffix editing, were calculated for both samples. The percent savings for majority and unanimously rated words in Sample A were 33 and 43% respectively. In Sample B the percent savings were 32 and 28%.

An analysis of the individual errors committed by SUFCUT (Table 4) indicates that all types of errors were distributed evenly throughout both rating categories. Some trends are apparent, however. For instance, it does appear that there were more omission errors than inclusive errors. The two types of omission errors, under edits and edit ignores, occurred three times as often as the inclusive-type error over edits. Two of the errors in the over-edits category were incurred when SUFCUT combined two words that didn't belong together--according to the raters. In one case, SUFCUT combined ACADEMIC and ACADEMY into the same root form, ACADEM.

Several other over edit errors occurred when SUFCUT inappropriately stripped S from such words as ALPS and ALWAYS. Although S appears frequently as a suffix for many nouns and verbs, there are a few instances in the English language when word roots end with S. Unfortunately, at this time SUFCUT has no way of determining when S is a legitimate suffix and when it is a part of the word root. Consequently, it removes indiscriminantly any S that appears at the end of a word. An exception to this operation is the editing of words ending with SS. The latter S in the SS string is not removed unless the SS combination is a substring of the NESS or LESS suffixes, in which case the entire string is removed.

Another type of under edit error committed by SUFCUT was the failure to strip away unusual suffix strings. Such strings as CE, EE, and WARD were not in SUFCUT's repetoire. As a result,

words like ABSENCE, ABSENTEE, and AFTERWARD were ignored by the program but edited by the raters. Errors specifically of this type could be prevented in the future by updating the program SUFTRE.

Chapter 6

CONCLUSIONS AND DISCUSSION

In summary, the purpose of this thesis was to present a computational method for trimming words down to their root form. A heuristic solution was proposed consisting of two types of heuristics--general and task-specific heuristics. The interchangeable structure of the task-specific heuristics promises numerous advantages. One suffix editor can be modified to apply to numerous heterogeneous types of problems. Suffixes stored on a binary tree data structure conserve 1) space by overlapping identical strings and 2) search time by varying the order of suffixes so that the most frequently occurring strings are searched first. The suffix tree also enables the interplay of powerful follow-up operations. These operations are initiated by exit values on the SUFTRE and enable such sophisticated manipulations as multiple edits and the creation of new roots by adding vowels.

In contrast with other suffix editing schemes, this procedure does not resort to a word-root dictionary. Consequently, no part of the execution time is consumed by costly dictionary look-up assignments. This procedure does require, however, that all words in the data base be sorted into alphabetical order. In most cases, this requirement does not pre-empt additional object time since many text processors which might be supervising SUFCUT will have already sorted the words as a

matter of standard practice.

When applied to English language data, SUFCUT had been found to function with a low to moderate rate of errors. Since the raters were inconsistent in editing 12-21% of the data base, it has not been possible to determine exactly how much of the total error was attributable exclusively to SUFCUT. Most of the program errors were omission errors. They occurred when the program failed to remove all the characters from the suffix portion of a word. Inclusion errors, such as inappropriately combining words with different word roots and trimming a word down to less than its proper word root, have been found to occur at a low frequency. Thus, the program has adopted a conservative mode of procedure. The fact that omission errors constituted a greater portion of the errors is an advantage to SUFCUT. Inclusion errors have a greater detrimental influence on the editing process, since different words are combined into the same category.

In comparing SUFCUT's performance with that of other editors, it is apparent that SUFCUT's percent of savings is below those figures which Haraway and Iker reported for their STRIP program. The percentages of savings for SUFCUT edited passages ranged from 28 to 43%, whereas Haraway and Iker (13:5-6) have reported that their program STRIP had reduced certain documents by almost 50%. The greater percentage of savings associated with the STRIP program might be due to two factors. First, STRIP implements both prefix and suffix-editing routines, thereby addressing itself to a broader range of conditions in the data.

Secondly, the types of samples on which both programs operated were different. STRIP was applied to the transcripts of psychotheraputic sessions of an itinerant salesman (12:142). SUFCUT, on the other hand, was run on data gathered principally from interviews with college students (14:572). The SUFCUT data probably demonstrated a broader distribution of word usage. For instance, the verbal output of several people usually exhibits a greater variety of word usage than that of a single person. Also the breadth of a salesman's vocabulary is probably less than that of the average college student. These factors may have interacted so that the data which the STRIP program processed were easier to edit since on the whole they consisted of fewer different words. Thus, variations in the data may have contributed heavily to the reported program differences.

The problem of the increased rate of erroneous edits of words in the majority category (rater agreement = 2/3) needs further research. If it were possible to attain unanimous rater agreement (3/3) on all words in a document, then a more accurate estimate of program error could be made. In the future the discrepancies among raters might be lessened by taking the following steps. Provide the rater with more information about each word. For instance, the sentence in which the word appeared might be presented along with the word. Furthermore, the rater might be allowed to read the entire document first and then make edits directly from the document. In this manner each word would be edited according to its meaning in its specific context. This

approach would prevent raters from supplying their own idiosyncratic meanings to words. This happens when raters are given simply a list of words. Some of the words on a list can have numerous meanings depending on their context. Thus, it is entirely possible that each rater could assume a different meaning for the same word.

Another means of lessening discrepancies is to reduce the heterogeneity among the raters. Future research should include raters who are experts in the area with which the document is concerned. Hopefully, these experts would rely on equivalent frames of reference while interpreting the meaning of words in the document.

As mentioned earlier, often the same word can have different meanings in separate contexts. For instance, the word LIKE, when used as a noun or a verb, usually expresses a feeling of positive attraction and affection; however, as a preposition and adjective, it generally expresses the state of having the same or nearly the same appearance, qualities, or characteristics. How much this type of variation in meaning affects the accuracy of a suffix editor has not been determined. If an editor generates the same root regardless of the whole word's context, then the processor is creating a certain amount of error. Is this error small enough so that it can be tolerated? Or should some type of corrective measures be taken?

One means of handling words whose meanings change with context is to label all words according to their parts of speech.

Thus, words from the same generic root but appearing as different parts of speech would be treated as separate word roots. However, this approach requires that a grammatical coder be inserted into the SUFCUT program. Whether this step is necessary depends upon the results of further research.

ACKNOWLEDGEMENTS

I am profoundly grateful to all those people who have helped me in any way during the preparation of this thesis. In particular, I would like to acknowledge the assistance provided by the members of my committee--Dr. Holly C. Fryer, Dr. Leon Rappoport, and Dr. William J. Conover. Their suggestions and recommendations have rendered major contributions to this effort. In addition, I am very grateful for the invaluable assistance of Dr. Paul Fisher. Dr. Fisher has unhesitatingly spent many hours helping me through numerous problems. His ideas have contributed generously to almost every aspect of this thesis.

I also wish to acknowledge the following people: Dr. David Danskin and Dr. Carroll Kennedy for granting me permission to sample from their data bank; Ann Scott, David Scott, and George Cvetkovich for serving as raters in the validation study; and Mary Hughes for typing this manuscript.

I would like to express my gratitude to the National Institute of Health for providing me with financial support during my program of study at Kansas State.

Above all, I am deeply indebted to my wife Carol. Her patience and encouragement have supported me throughout.

REFERENCES

1. Berns, Gerald M. "Description of FORMAT, a Text-Processing Program," Communications of the Association For Computing Machinery. 1969, 12:3, 141-146.

2. Brooks, Frederich P. and Iverson, Kenneth E. Automatic Data Processing, New York: John Wiley, 1963.

3. Borko, Harold A. "Automatic Document Classification," Journal of the Association For Computing Machinery. 1963, 70, 151-162.

4. _____. "A Factor-Analytically Derived Classification System for Psychological Reports," Perceptual and Motor Skills. 1965, 20, 393-406.

5. _____ and Bernick, Myrna. "Automatic Document Classification. Part II. Additional Experiments," Journal of the Association For Computing Machinery, 1964, 11, 138-151.

6. Brown, Roger. Social Psychology. New York: The Free Press, 1967.

7. Dixon, Theodore R. and Horton, Daniel L. Verbal Behavior and General Behavior Theory. Englewood Cliffs: Prentice-Hall, 1968.

8. Earl, Lois L. "Structural Definitions of Affixes from Multisyllable Words," Mechanical Translation and Computational Linguistics, 1966, 9:2, 34-37.

9. _____. "Part-of-Speech Implications of Affixes," Mechanical Translation and Computational Linguistics. 1966, 9:2, 38-43.

10. _____. "Automatic Determination of Parts of Speech of English Words," Mechanical Translation and Computational Linguistics, 1967, 10:3-4, 53-67.

11. Harman, Harry H. Modern Factor Analysis, Chicago: University of Chicago Press, 1967.

12. Haraway, Norman I and Iker, Howard P. "Objective Content Analysis of Psychotherapy by Computer." Paper presented at 1964 Rochester Conference on Data Acquisition and Processing in Biology and Medicine, July, 1964, Rochester, N.Y.

13. _____. "Content Analysis and Psychotherapy," University of Pittsburgh. (Mimeographed).

14. Howes, Davis. "A Word Count of Spoken English," Journal of Verbal Learning and Verbal Behavior. 1966, 5, 572-604.

15. Klein, Sheldon and Sivamons, Robert F. "A Computational Approach to Grammatical Coding of English Words," Journal of the Association For Computing Machinery. 1963, 10:3, 334-347.

16. Knuth, Donald. Fundamental Algorethms: The Art of Computer Programming Volume I. Reading, Massachusetts: Addism-Wesley: 1968.

17. Maurer, Ward D. Programming: An Introduction to Computer Languages and Techniques. San Francisco: Holden-Day, 1968.

18. Pylyshyn, Zenon W. "FINDSIT: A Computer Program for Language Research," Behavioral Science. 1969, 14, 248-251.

19. Starkweather, John A. "Computer Methods for the Study of Psychiatric Interviews," Comprehensive Psychiatry, 1967, 11:6, 509-520.

20. _____ and Dicker, J. Barry. "Computer Analysis of Interview Content," Psychological Reports, 1964, 15, 875-880.

21. Stone, Philip J. "An Interactive Inquirer," Proceedings of the 1966 Invitational Conference on Testing Problems. 1967, 63-67.

22. _____ and Cambridge Computer Associates, Inc. User's Manual For the General Inquirer. Cambridge, Massachussetts: The M.I.T. Press: 1968.

23. _____ and others. The General Inquirer: A Computer Approach to Content Analysis. Cambridge, Mass.: The M.I.T. Press, 1966.

24. Storux, G. G., Naznitsky, I., and Rush, J. E. "Procedures for Converting Systematic Names of Organic Compounds into Atom-Bond Connection Tables," Journal of Chemical Documentation. 1967, 7:3, 165.

25. Veldman, Donald J. FORTRAN Programming for the Behavioral Sciences. New York: Holt, Rinehart, and Winston, 1957.

26. _____, "Computer-Based Sentence-Completion Interviews," Journal of Counseling Psychology. 1967, 14:2, 153-157.

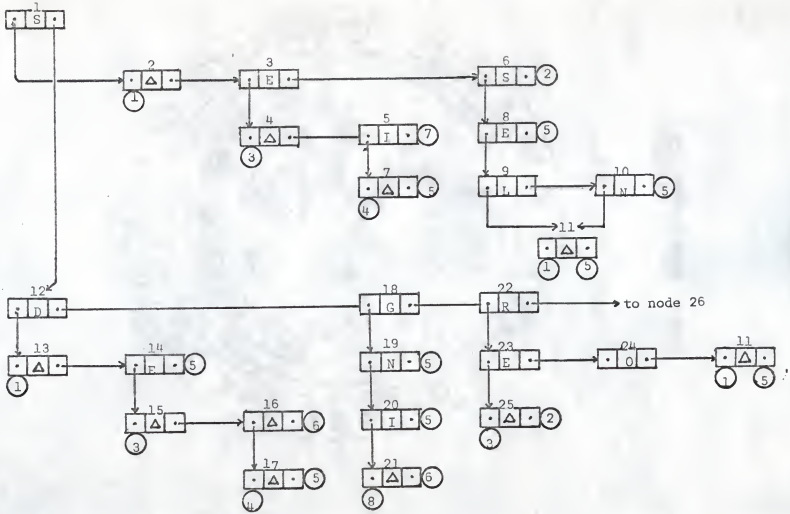
27. _____. "Computer Scoring of Sentence Completion Data," Behavioral Science. In Press.

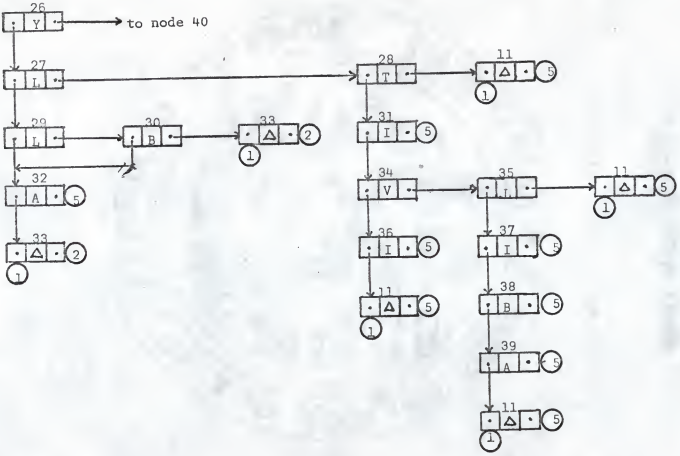
APPENDIX 1

A COMPLETE LISTING OF ALL SUFFIXES WHICH HAVE BEEN INCLUDED IN THE SUFTRE OF THE SUFCUT PROGRAM

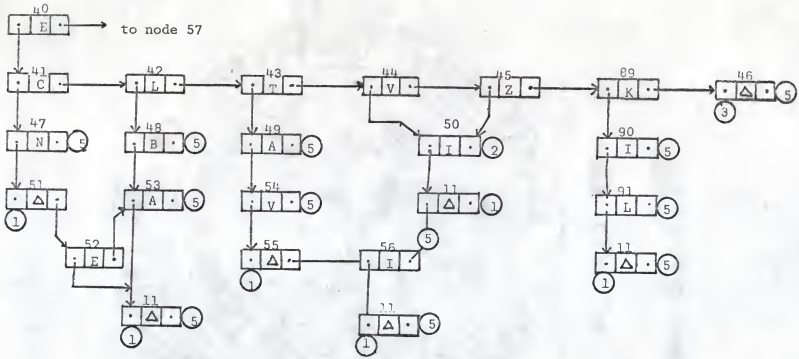
I C	I N G	S
I S T I C	I S H	E S
D	A L	I E S
E D	F O R M	L E S S
I E D		N E S S
E	N	S T
I V E	I O N	E S T
I Z E	R E N	I E S T
N C E	T I O N	M E N T
A B L E	A T I O N	Y
E N C E	I T I O N	L Y
L I K E	I Z A T I O N	I T Y
V A T E	E R	A L L Y
I V A T E	O R	A B L Y
		I V I T Y
		A B I L I T Y

Appendix 2. The Entire Suffix Tree (SUFTRE) of the SUFCUT Program

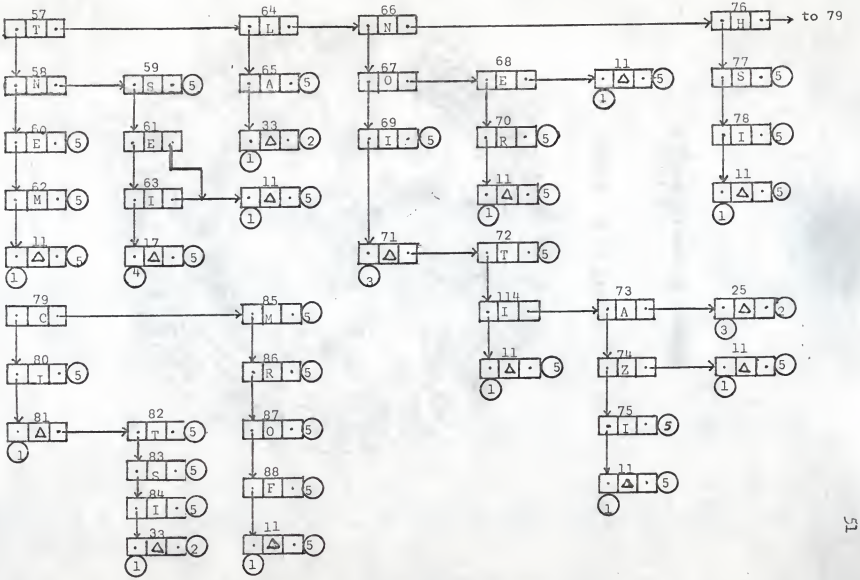



26
Y
to node 40
27
L
28
T
11
Δ
5
1
29
L
30
B
33
Δ
2
1
31
I
5
32
A
5
34
V
35
L
11
Δ
5
1
33
Δ
2
1
36
I
5
37
I
5
11
Δ
5
1
38
B
5
39
A
5
11
Δ
5
1

Appendix 2 (Continued)



Appendix 2 (Continued)



APPENDIX 3

THE SOURCE LISTING OF THE SUFCUT PROGRAM

```
C . . . . . S T E P 1 . . . . . .
      DIMENSION   INPUT(70)
      INTEGER     ALPH(26), BLANK,FREQ(1000),PTR(1000),WORD(1000,15)
C   INITIALIZE DATA.
      DATA BLANK,ALPH/' ','A','B','C','D','E','F','G','H','I','J','K',
     1 'L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z'/
      IPUT70 = BLANK
      I = 1
      NWORDS = 0
      DO 10 M=1,1000
      DO 10 N=1,15
   10 WORD(M,N) = 0
C   READ ONE CARD OF TEXT.
   11 READ (1,5,END=17) INPUT
    5 FORMAT(80A1)
C   BEGIN A COLUMN BY COLUMN ANALYSIS OF ONE CARD.   LET INPUT(J) REPRESENT
C   THE J TH CHARACTER FOUND IN THE JTH COLUMN OF THAT CARD.
      DO 16 J=1,70
C   TEST FOR A BLANK.
      IF (INPUT(J).EQ.BLANK) GO TO 13
C   CHARACTER IS NOT A BLANK.   IS IT A LETTER IN THE ALPHABET?
      DO 12 K=1,26
      KK = K
      IF (INPUT(J).EQ.ALPH(K)) GO TO 15
   12 CONTINUE
      GO TO 16
C   IF THE FIRST COLUMN ON A CARD IS A BLANK AND THE LAST COLUMN OF THE PRECED-
C    ING CARD IS ALSO A BLANK, THEN BRANCH AND BEGIN PROCESSING THE NEXT CHAR.
   13 IF (J.EQ.1.AND.IPUT80.EQ.BLANK)  GO TO 16
C   BUT IF THE LAST COLUMN OF THE PRECEDING CARD IS NOT A BLANK, THEN INCREMENT.
      IF (J.EQ.1)  GO TO 14
      IF (INPUT(J-1).EQ.BLANK)  GO TO 16
C   INCREMENT WORD COUNTER (IWORD) AND RESET THE LETTER POSITION POINTER (PTR).
```

APPENDIX 3

THE SOURCE LISTING OF THE SUFCUT PROGRAM

```
C . . . . . S T E P 1 . . . . . .
      DIMENSION   INPUT(70)
      INTEGER     ALPH(26), BLANK,FREQ(1000),PTR(1000),WORD(1000,15)
C   INITIALIZE DATA.
      DATA BLANK,ALPH/' ','A','B','C','D','E','F','G','H','I','J','K',
     1 'L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z'/
      IPUT70 = BLANK
      I = 1
      NWORDS = 0
      DO 10 M=1,1000
      DO 10 N=1,15
   10 WORD(M,N) = 0
C   READ ONE CARD OF TEXT.
   11 READ (1,5,END=17) INPUT
    5 FORMAT(80A1)
C   BEGIN A COLUMN BY COLUMN ANALYSIS OF ONE CARD.   LET INPUT(J) REPRESENT
C   THE J TH CHARACTER FOUND IN THE JTH COLUMN OF THAT CARD.
      DO 16 J=1,70
C   TEST FOR A BLANK.
      IF (INPUT(J).EQ.BLANK) GO TO 13
C   CHARACTER IS NOT A BLANK.   IS IT A LETTER IN THE ALPHABET?
      DO 12 K=1,26
      KK = K
      IF (INPUT(J).EQ.ALPH(K)) GO TO 15
   12 CONTINUE
      GO TO 16
C   IF THE FIRST COLUMN ON A CARD IS A BLANK AND THE LAST COLUMN OF THE PRECED-
C    ING CARD IS ALSO A BLANK, THEN BRANCH AND BEGIN PROCESSING THE NEXT CHAR.
   13 IF (J.EQ.1.AND.IPUT80.EQ.BLANK)  GO TO 16
C   BUT IF THE LAST COLUMN OF THE PRECEDING CARD IS NOT A BLANK, THEN INCREMENT.
      IF (J.EQ.1)  GO TO 14
      IF (INPUT(J-1).EQ.BLANK)  GO TO 16
C   INCREMENT WORD COUNTER (IWORD) AND RESET THE LETTER POSITION POINTER (PTR).
```

```
   14 NWORDS = NWORDS + 1
      I = 1
      GO TO 16
   15 WORD(NWORDS+1,I) = KK
      I = I + 1
      IF (I.GT.15)  I=15
   16 CONTINUE
C    SAVE THE CHARACTER IN COLUMN 80 BEFORE READING IN A NEW CARD.
      IPUT70 = INPUT(70)
      GO TO 11

C . . . . . S T E P 2 . . . . . .

C
C
C    SORT THE WORD ARRAY.
C
C
C    AFTER ALL ELEMENTS IN THE WORD ARRAY HAVE BEEN SORTED INTO ALPHABETICAL
C    ORDER, SET FREQUENCY COUNTERS (FREQ) AND THE POINTERS (PTR).
   17 DO 18 I=1,NWORDS
      FREQ(I) = 1
   18 PTR(I) = I+ 1
      PTR(NWORDS+1) = 0

C . . . . . S T E P 3 . . . . . .

      I = 1
      J = PTR(I)
   31 DO 32 K=1,15
      IF (WORD(I,K).NE.WORD(J,K)) CALL SUB1(I,J,K,FREQ,PTR,WORD,&35,&36)
```

```
   32 CONTINUE
C   WORD(I) = WORD(J), COMBINE FREQUENCIES AND DELETE WORD(J).
      FREQ(I) = FREQ(I) + FREQ(J)
      JJ = J
   33 IF(PTR(JJ-1).EQ.J) GO TO 34
      JJ=JJ-1
      GO TO 33
   34 PTR(JJ-1) = PTR(J)
      PTR(J) = -1
      J = JJ - 1
   35 IF (PTR(J).EQ.0) GO TO 39
      J = PTR(J)
      GO TO 31
   36 IF (I.EQ.0)  GO TO 37
      IF (PTR(I).EQ.0 )  GO TO 37
      J = PTR(PTR(I) )
   39 I = PTR(I)
      IF (I.EQ.0)  GO TO 37
      IF (PTR(I).EQ.0)  GO TO 37
      GO TO 31
   37 CONTINUE


C . . . . . S T E P 4 . . . . . .


      I = 1
      ICOUNT = 0
C   PRINT OUT A HEADING FOR THE RESULTS.
      WRITE (3,6)  NWORDS
    6 FORMAT ('1', 5X, ' AN ALPHABETICAL LISTING OF ALL WORD ROOTS FOUND IN THIS
     1 IN THIS DOCUMENT  (ORIGINAL NUMBER OF WORDS =  ', 13,' )'////)
   41 IF (PTR(I).EQ.0)  GO TO 44
      DO 42  N=1,15
   42 INPUT(N) = BLANK
```

```
      DO 45 J=1,15
      IF (WORD(I,J).EQ.0)  GO TO 46
      DO 43 M=1,26
      INPUT(J) = ALPH(M)
      IF (WORD(I,J).EQ. M )  GO TO 45
   43 CONTINUE
   45 CONTINUE
   46 ICOUNT = ICOUNT + 1
      WRITE(3,7) ICOUNT, (INPUT(J),J=1,15), FREQ(I)
    7 FORMAT (' ', I3, 9X, 15A1, 5X, I5)
      I = PTR(I)
      GO TO 41
   44 WRITE (3,8)
    8 FORMAT ( 'l', 'PROBLEM COMPLETED;  NORMAL TERMINALION')
      STOP
      END

      SUBROUTINE SUB1    (I,J,K,FREQ,PTR,WORD,*,*)
      INTEGER  FREQ(1000),PTR(1000),STACK1(5,12),STACK2(5,12),WORD(1000,15)
     1 15)
C   INITIALIZE
      DO 1 M=1,5
      DO 1 N=1,12
      STACK1(M,N) = 0
    1 STACK2(M,N) = 0
      I1 = 1
       I2 = 0
      J1 = 1
      J2 = 0
      IF (WORD(I,K).NE.0) GO TO 2
       I2 = I2 + 1
       GO TO 5
    2 DO 3 N=K,14
      NN = N
       IF (WORD(I,N+1).EQ.0)  GO TO 4
```

```
 3 CONTINUE
   NN =15
 4 L1 = NN
   CALL SUB2      (I,I2,K,L1,STACK1,WORD,&13)
 5 IF (WORD(J,K).NE.0) GO TO 6
   J2 = J2 + 1
   GO TO 9
 6 DO 7 N=K,14
   NN = N
   IF (WORD(J,N+1).EQ.0) GO TO 8
 7 CONTINUE
   NN = 15
 8 L2 = NN
   CALL SUB2     (J,J2,K,L2,STACK2,WORD,&13)
   IF (12.EQ.0.OR.J2.EQ.0)  RETURN1
 9 CALL SUB3 (I,J,I1,I2,J1,J2,FREQ,PTR,STACK1,STACK2,&12,&10)
10 N = 0
   DO 11 M=K,12
   N = N + 1
11 WORD(I,M) = STACK1(I1,N)
12 RETURN1
13 RETURN2
   END

   SUBROUTINE  SUB2  (IWORD,I2,K,L,STACK,WORD,*)
   INTEGER     EXITVL,P,LPTR(114),RPTR(114),STACK(5,12),SUFTST(114),
  1  WORD(1000,15)
   DATA  SUFTST          /19,0,5,0,9,19,0,5,12,14,0,4,0,5,0,9,0,7,14,
 1 9,0,18,5,15,0,25,12,20,12,2,9,1,0,22,12,9,9,2,1,5,3,12,20,22,26,0
 2 ,14,2,1,9,0,5,1,22,0,9,20,14,19,5,5,13, 9,12,1,14,15,5,9,18,0,20,
 3 1,26,9,8,19,9,3,9,0,20,19,9,13,18,15,6, 11, 9,12,19, 5, 9,19, 5,
 4 12,14,7,14,9,20,19,5,4,5,25,12, 5,11, 9,12,0,9/
   DATA   LPTR /2,-1,4,-3,7,8,-4,9,11,11,-1,13,-1,15,
  5  -3,17,-4, 19,20,21,-3,23,25,25,-3,27,29,31,32,32,34,33,-1,36,37,
```

```
     6  11,38,39,11,41,47,48,49,50,50,-3,51,53,54,11,-1,11,11,55,-1,11,
     7  58,60,61,62,63,11,17,65,33,67,69,70,71,11,-3,114,74,75,11,77,78,
     8  11,80,81,-1,83,84,33,86,87,88,11,  90,91,11,93,94,-4,96,97,-1,
    9-1,100,101,-8,103,104,-1,106,-9,108,-1,110,111,112, -1,-2,11/
      DATA   RPTR / 12,3,6,5,-7,-2,-5,-5,10,-5,-5,18,14,-5,16,
     A  -6,-5,22,-5,-5,-6,26,24,11,-2,40,28,33,30,33,-5,-5,-2,35,11,-5,
     B  -5,-5,85,57,42,43,44,45,89,-5,-5,-5,-5,-2,52,53,-5,-5,56,-5,64,
     C  59,-5,-5,11,-5,11,66,-5,76,68,11,-5,-5,72,-2,25,11,-5,79,-5,-5,
     D  -5,-5,82,-5,-5,-5,-5,-5,-5,-5,46,-5,-5,99,95,-7,-7,-5,98,-5,102,
     E -5,-5,105,-5,-5,107,-5,109,-5,-5,-5,-5,-5,15,73/
      P = 1
      LL = L
      IF (K.LE.3)  GO TO 201
   1  IF (WORD(IWORD,L).EQ.SUFTST(P) ) GO TO 2
      P = RPTR(P)
      GO TO 4
   2  P = LPTR(P)
   3  L = L-1
   4  IF (P.LT.0)  GO TO 5
      IF (L.EQ.K-1) GO TO 14
      GO TO 1
   5  EXITVL = -P
      GO TO  (6,6,10,10,12,13,60),  EXITVL
  60  L = LL - 1
   6  III = 0
      DO 7 II=K,L
      III = III + 1
   7  STACK(I2+1,III) = WORD(IWORD,II)
       I2 = I2 + 1
      GO TO (8,9,8,8,8,8,9),  EXITVL
   8  RETURN
   9  P = 1
      LL = L
      GO TO 1
  10  III = 0
```

```
      DO 11 II=K,L
      III = III+ 1
   11 STACK(I2+1,III) = WORD(IWORD,II)
      IF (EXITVL.EQ.3)  STACK(I2+1,III   ) = 5
      IF (EXITVL.EQ.4) STACK(I2+1,III   ) = 25
      I2 = I2 + 1
      GO TO 6
   12 III = 0
      DO 16  II = K,LL
      III = III + 1
   16 STACK(I2+1,III) = WORD(IWORD,II)
      I2 = I2 + 1
      RETURN
   13 IF (L.NE.K)  GO TO 9
      IF (WORD(IWORD,K-1).NE.WORD(IWORD,L)) GO TO 8
      I2 = I2 + 1
      RETURN
   14 IF (SUFTST(P).EQ.0)  GO TO 15
      P = RPTR (P)
      IF (P.LT.0)  RETURN
      GO TO 14
   15 EXITVL = -LPTR(P)
      GO TO (19,19,17,18,19,19,19),  EXITVL
   17 STACK(I2+1,1)  = 5
       I2 = I2 + 2
      RETURN
   18 STACK(I2+1,1) = 25
      I2 = I2 +1
   19 I2 = I2 + 1
      RETURN
   20 LL = L
  201 P = 92
   21 IF (WORD(IWORD,L).EQ.SUFTST(P) ) GO TO 22
      P = RPTR(P)
      IF (P.LT.0)  GO TO 25
```

```
      GO TO 21
   22 P = LPTR(P)
   23 IF (P.LT.0)  GO TO 25
   24 L = L-1
      IF (L.EQ.3)  GO TO 22
      IF (L.EQ.2)  RETURN1
      GO TO 21
   25 EXITVL = -P
      GO TO (26,26,40,26,40,30,28,30,30,30) ,  EXITVL
   31 L = L - 1
   26 DO 27 M=L,LL
   27 WORD(IWORD,M) = 0
      IF (EXITVL.EQ.2)  GO TO 20
      IF(EXITVL.EQ.4)  WORD(IWORD,L) = 25
   40 RETURN1
   28 WORD(IWORD,LL) = 0
      L = L + 1
      GO TO 20
   30 IF (WORD(IWORD,L-1).EQ.WORD(IWORD,L-2))  GO TO 31
      IF(EXITVL.EQ.9)  L = LL
      GO TO 26
      END

      SUBROUTINE  SUB3 (I,J,I1,I2,J1,J2,FREQ,PTR,STACK1,STACK2,*,*)
      INTEGER  FREQ(1000),PTR(1000),STACK1(5,12),STACK2(5,12)
    1 DO 2 N=1,12
      NN = N
      IF ((STACK2(J1,NO-STACK1(I1,NO ).EQ.0) GO TO 2
      GO TO 5
    2 CONTINUE
C     WORD ROOTS ARE EQUIVALENT,  COMBINE WORD CATEGORIES.
      FREQ(I) = FREQ(I) + FREQ(J)
      JJ = J
    3 IF (PTR(JJ-1) .EQ. J )   GO TO 4
      JJ = JJ - 1
```

```
      GO TO 3
    4 PTR(JJ-1) = PTR(J)
      PTR(J) = -1
      J = JJ - 1
      RETURN2
    5 IF ((STACK2(J1,NN)-STACK1(I1,NN)).LT.0) GO TO 6
      J1 = J1 + 1
      IF (J1.EQ.(J2+1))  RETURN1
      GO TO 1
    6 I1 = I1 + 1
      IF (I1.EQ.(I2+1)) RETURN1
      GO TO 1
      END
```

APPENDIX 4

DIRECTIONS FOR THE WORD ROOT IDENTIFICATION TASK

Directions: I am interested in finding out what you believe are the word roots of the words listed on the attached answer forms. A word root is the smallest portion of a word that contains the same general meaning as the whole word. In many cases, the root of a word is the entire word. For example, the words HAT, CAR, PUNISH, and PROPERTY are word roots. They cannot be reduced any further without losing their original meaning. In other cases, a whole word can be reduced to a smaller root word by removing one or more suffixes. The words ACTS, ACTING, ACTION, and ACTIVATED all can be reduced to the same word root ACT. Furthermore, there are a few special words which are reduced by first removing a suffix and then adding an "E" or "Y". In this manner, CONCENTRATION and TRIES can be trimmed down to CONCENTRATE and TRY.

I am going to give you a list of 200 words. What I would like you to do is to circle that part of each word which you think is the root of that word. Remember the root is the smallest possible word within the whole word with the same general meaning as the whole word. Feel free to add letters to a stem to form a new root, if you believe it is necessary. There is no time limit and you may use a dictionary.

A COMPUTATIONAL METHOD FOR REDUCING WORDS TO THEIR GENERIC ROOTS UTILIZING A SUFFIX-EDITING ROUTINE

by

MARTIN LEON ZOLA

B. A., University of Notre Dame, 1967

AN ABSTRACT OF A MASTER'S THESIS

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Statistics and Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1969

Computational methods of suffix editing play a major role in a large number of language processing programs. The authors of these programs--concerning themselves primarily with their main procedures--have provided little information on the performance of their editors. Practically all current editors consist of two parts: a set of suffixes to be edited and a dictionary of word roots. A strip-match technique is employed whereby each word is stripped of its suffixes and a dictionary is searched for a matching root.

A new approach to the problem of suffix editing has been presented here. It consists entirely of operations on words without recourse to a word-root dictionary. This approach is based on two types of heuristics. A set of general heuristics provide a style for approaching numerous language manipulation tasks. The second set, task-specific heuristics, are detailed rules which are designed to be applied to only one problem. One type of task-specific heuristic of particular importance is the follow-up command. A follow-up command is designated for each suffix string. When a suffix is removed, its corresponding follow-up command is enacted.

A FORTRAN program based upon these heuristics was developed to perform suffix editing of words in English language documents. It contains an interchangeable component which may be modified in order to adapt it to other language processing problems.

This program has been run on three samples of text taken

from large English language documents. The program's performance was evaluated by comparing its output with the protocols of three human editors who processed the same data manually. The percentage of erroneous edits varied from 2-32% per sample depending upon the criterion used. As the rater agreement of the words in the criterion dropped, the percentage of program errors increased. The words with multiple suffixes caused the most difficulty for both the program and the raters. Suggestions for further research were discussed.