

Simons

WORKING PAPERS FOR THE LANGUAGE VARIATION  
AND LIMITS TO COMMUNICATION PROJECT

Number 7

A PACKAGE OF COMPUTER PROGRAMS FOR  
THE ANALYSIS OF LANGUAGE SURVEY WORD LISTS

Gary Simons  
Cornell University  
and  
Summer Institute of Linguistics

1977

A PACKAGE OF COMPUTER PROGRAMS FOR  
THE ANALYSIS OF LANGUAGE SURVEY WORD LISTS

Gary Simons

Cornell University  
and  
Summer Institute of Linguistics

0. INTRODUCTION

1. AN OVERVIEW OF THE WORD LIST ANALYSIS PACKAGE

- 1.1 The "partial automation" and "interactive" approaches
- 1.2 The overview

2. A USER'S GUIDE TO THE WORD LIST ANALYSIS PACKAGE

2.1 Input

- 2.1.1 Enter a gloss file
- 2.1.2 Enter the first word list
- 2.1.3 Extract the word list alone
- 2.1.4 Phonemic analysis
- 2.1.5 Entering additional word lists
- 2.1.6 Edit the previous word list
- 2.1.7 Edit a word list saved on tape
- 2.1.8 Enter the whole list into a clean buffer
- 2.1.9 Merge word lists into a single master file

2.2 Phonemic analysis

- 2.2.1 Preparation
- 2.2.2 Phone occurrences
- 2.2.3 Phone co-occurrences
- 2.2.4 Word shapes
- 2.2.5 Syllable shapes

2.3 Lexicostatistic analysis

- 2.3.1 Count cognates
- 2.3.2 Build a matrix of cognate counts
- 2.3.3 Permute the matrix of cognate counts

2.4 Lexical isogloss analysis

- 2.4.1 Extract a lexical isogloss file
- 2.4.2 Analysis of lexical isoglosses

2.5 Phonostatistic analysis

- 2.5.1 Pre-program data formatting
- 2.5.2 Format checking
- 2.5.3 Enter the degrees of difference table
- 2.5.4 Phonostatistics

2.6 Comparative method

- 2.6.1 Pre-program data formatting
- 2.6.2 Count correspondence sets
- 2.6.3 Phonological isogloss analysis
- 2.7 Refined phonostatistic analysis

### 3. THE PTP PROGRAMMING LANGUAGE

#### 3.1 Updates to the PTP user's manual

#### 3.2 A summary of PTP commands and special characters

### 4. PROGRAM LISTINGS

#### 4.1 Input

##### 4.1.1 Copy 2 to 3

##### 4.1.2 Extract list

##### 4.1.3 Zero 3

##### 4.1.4 Merge glosses into list

##### 4.1.5 Merge lists

#### 4.2 Phonemic analysis

##### 4.2.1 Words alone

##### 4.2.2 Phone occurrences

##### 4.2.3 Phone co-occurrences

##### 4.2.4 Word shapes

##### 4.2.5 Syllable shapes

##### 4.2.6 Find examples

#### 4.3 Lexicostatistic analysis

##### 4.3.1 Count cognates

##### 4.3.2 Build cognate matrix

##### 4.3.3 Permute matrix

#### 4.4 Lexical isogloss analysis

##### 4.4.1 Lexical isoglosses

##### 4.4.2 Isogloss analysis

#### 4.5 Phonostatistic analysis

##### 4.5.1 Check format

##### 4.5.2 Phonostatistics

#### 4.6 Comparative method

##### 4.6.1 Prepare a set

##### 4.6.2 Delete a set

##### 4.6.3 Copy a set

##### 4.6.4 Count correspondence sets

##### 4.6.5 Isogloss analysis

##### 4.6.6 Find examples

#### 4.7 Refined phonostatistic analysis

### REFERENCES

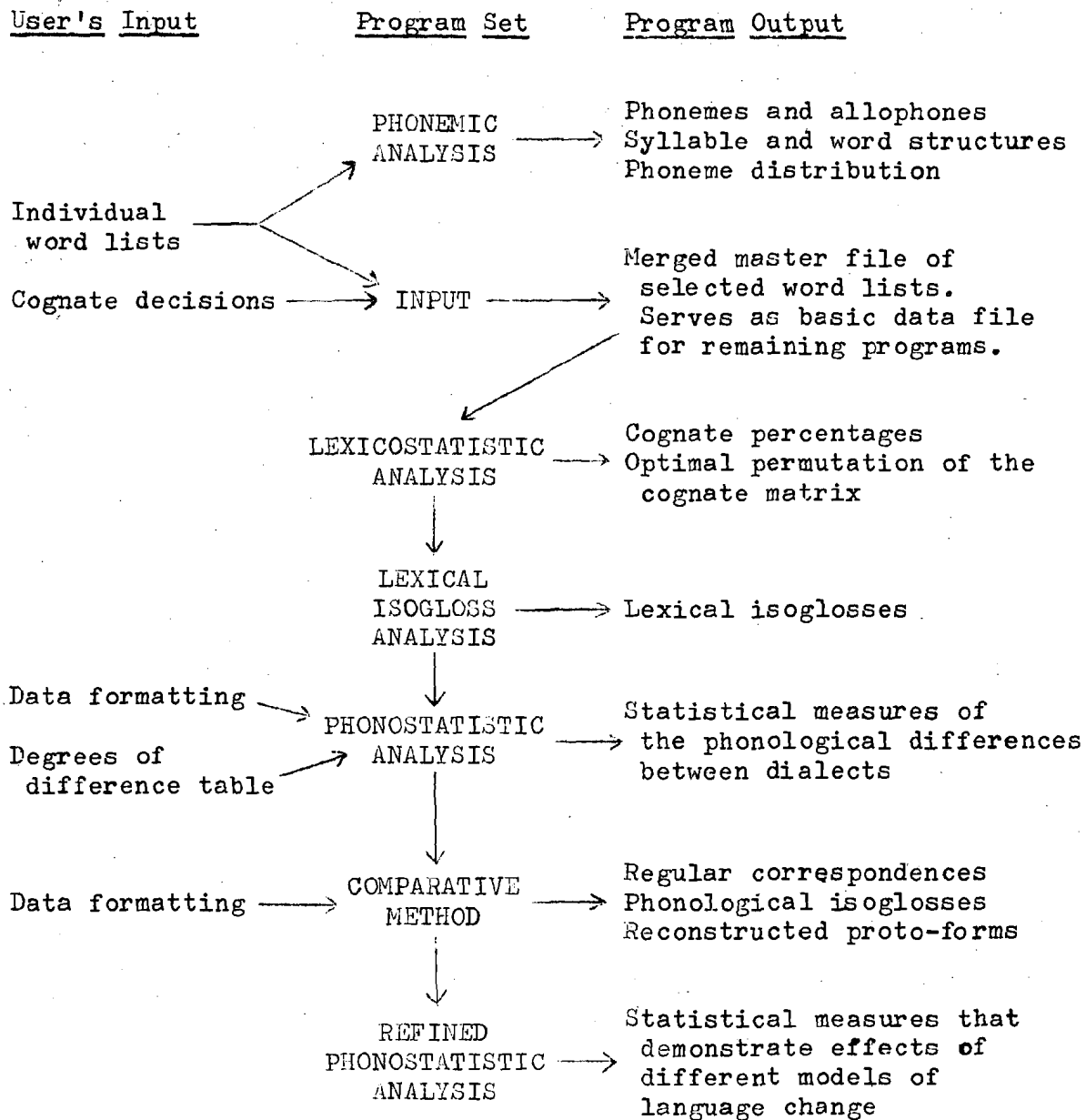
## O. INTRODUCTION

The Word List Analysis Package is a package of computer programs which aid the linguist in his analysis of the word list data collected in a language survey. The phases of the analysis in which the programs assist the linguist are phonemic analysis, lexicostatistics, lexical isogloss analysis, phonostatistics, comparative method, phonological isogloss analysis, and a refined phonostatistics in which the relative effects of different causes of sound change can be quantified. The Word List Analysis Package runs on a fully portable microcomputer which will operate in field conditions. Use of these tools can open up to the linguist in the field a depth of analysis which is seldom achieved.

Section one of the paper gives a brief overview of the Word List Analysis Package and discusses the "partial automation" and "interactive" approaches taken in writing the programs. Section two is a detailed user's guide to the package of programs. It is designed in such a way that the linguist with no computer background, after a minimum of instruction on operating the microcomputer, should be able to follow the guide step by step to perform a full analysis of his word list data. Section three gives a summary of the programming language called PTP, the Programmable Text Processor, which was used to write the programs. Special attention is given to the changes that have been made to PTP since the PTP user's manual (Simons 1977) was written. In section four, complete listings and structured descriptions of all the programs comprising the Word List Analysis Package are given.

The Word List Analysis Package was developed in conjunction with the Cornell University research project "Language Variation and Limits to Communication" (Dr. Joseph E. Grimes, Principal Investigator). The project is being partially supported by grant BNS76-06031 from the National Science Foundation. The programs are currently operating on a portable microcomputer called the Electronic Text Processor, model ETP-8L. It is an Intel 8080 based 8-bit microcomputer with a 16 kilobyte (roughly 16,000 characters) memory. During 1977 the author has used an ETP-8L in the Solomon Islands to process the data resulting from linguistic field research.

Figure 1. Overview of the Word List Analysis Package



## 1. AN OVERVIEW OF THE WORD LIST ANALYSIS PACKAGE

An overview of the whole Word List Analysis Package is diagrammed in Figure 1. The figure is set out in three columns. The first column, labelled "User's Input", shows what data and formatting must be entered by the user before the programs can be run. The second column, labelled "Program Set", lists the seven different sets of programs of which the whole package is comprised. The third column, labelled "Program Output", lists the kinds of output which the user can obtain from the results of the programs.

### 1.1 The "partial automation" and "interactive" approaches

The approach taken in these programs could be called one of "partial automation". That is, the derivation of the results listed in the third column of Figure 1 is not entirely automatic. The computer is programmed to do those things which it does well-- sorting, counting, comparing, filing, permuting, finding examples. The user for his part makes the key decisions, basing those decisions on both the computer's tabulations and his own experience. Thus when Figure 1 states that "phonemes and allophones" are outputs of the phonemic analysis set, it means that the phone occurrence, phone co-occurrence, and phone distribution counts output by the phonemic analysis programs are the results to which the user can apply his linguistic experience to obtain directly an analysis of the phonemes and their allophones.

The "partial automation" approach was taken by Frantz (1970) in his COMPASS program for assisting the comparative linguist. In many ways, my treatment of phonostatistics and the comparative method are based on his approach. In that approach the linguist determines which words are probable cognates and which segments in those words correspond. The computer then performs the time consuming intermediate step of comparing the cognate forms to compile statistics on the correspondence sets that occur and lists of supporting evidence for each. The linguist, by comparing the statistics and by examining the forms which evidence the correspondence sets, makes the final decisions as to which are the regular correspondences and which are the irregular.

In many cases the user's decision making is further aided by an "interactive" approach. There are a number of programs which reorder the results and display the same facts in a different way. Other programs find all the examples of a specific thing. In running these programs the user is able to "interact" with the computer by using it to test different hypotheses and find examples which support or refute those hypotheses. This is another way in which the computer aids the linguist in making final decisions.

To fully automate the decision making processes (that is, to embody the complete experience of the linguist in a computer program) is probably impossible. In the least it would tax the upper limits of the microcomputer and the programmer. Thus we adopt a "partial automation" approach in which the computer, by doing what it can do best, frees the linguist to devote his time to doing what he can do best.

## 1.2 The overview

The Word List Analysis Package begins with a set of INPUT programs by means of which the user enters all of the word lists which are to be analyzed. At the same time the user enters the cognate set identifiers that indicate which words are judged to be cognate and which words are not. As an individual word list is entered it may be run through the PHONEMIC ANALYSIS programs in order to determine for that dialect the phonemes and their allophones, the distribution of phonemes, and the structure of syllables and words. After all the word lists have been entered and saved on tape, the user selects a set of lists (either all of them, or any subset) which he wishes to analyze. These individual lists are merged into a single master file which serves as the basic data file for the remaining programs.

In the LEXICOSTATISTIC ANALYSIS programs, the number of cognate words between each pair of lists in the merged master file is counted. The cognate counts are built into a matrix which can be interactively permuted in order to explore the divergence and convergence relationships between languages and to find the optimal permutation of the matrix for display purposes.

The LEXICAL ISOGLOSS ANALYSIS programs first extract a lexical isogloss file from the merged master file. Then an interactive program allows the user to permute and re-sort the isogloss file in order to group together word list items with like isogloss patterns. When the grouping of isogloss patterns is completed, the isoglosses can be copied directly onto a map as isogloss lines. Each line can be labelled with the numbers of the items exemplifying the isogloss.

Before the PHONOSTATISTIC ANALYSIS programs can be run, the user must do some manual formatting of the merged master file to ensure that all corresponding segments in cognate words are in the same position in the word. A program checks that the formatting is correct before moving to the analysis step. The checking program also extracts a list of all the sound correspondences that occur in the data. Before proceeding to the

phonostatistic analysis, the user is required to assign a degree of phonological difference to each sound correspondence and enter it into the table of correspondences. The phonostatistics program is then run to compute statistical measures of the phonological difference between each pair of dialects.

Before the COMPARATIVE METHOD analysis can be carried out, it is necessary for the user to do some additional formatting of the merged word lists. Three programs are provided to help with this. When formatting is complete, a program is run to make a tabulation of all the correspondence sets that occur and their frequency of occurrence. Two programs are provided to help the user in his analysis of the results--a program which permutes and re-sorts the table of correspondence sets in order to group like correspondences, and a program which finds all the examples of any given correspondence set. On the basis of this analysis, the user should be able to posit the regular sound correspondences between the dialects represented in the word lists. On the basis of the regular sound correspondences he can define phonological isoglosses and reconstruct proto-forms.

The REFINED PHONOSTATISTIC ANALYSIS combines phonostatistic analysis with the results of the comparative method to break down the analysis of degrees of difference into the degrees of difference accounted for by the regular sound correspondences between dialects and the degrees of difference accounted for by irregular correspondences. The results of this analysis provide statistical measures of the relative effect of different models of language change in explaining the phonological relations between languages. For instance, the degrees of difference accounted for by regular correspondences are explained by a divergence model (eg. sound drift, sound laws with no exceptions). Degrees of difference accounted for by irregular correspondences give evidence for social explanations of language change (eg. word tabooing, convergence via contact).

## 2. A USER'S GUIDE TO THE WORD LIST ANALYSIS PACKAGE

The user's guide is set out in a step-by-step instruction format. Unless otherwise noted, the user is meant to proceed sequentially from section to section. If the sequence of steps is other than to the following section, a statement telling which section to go to next will be given. Whenever the instruction to run a program is given, a cross-reference to that program's listing in section four is also given.



Before the user can operate the Word List Analysis Package on the microcomputer, it is necessary that he be able to operate the immediate mode commands of PTP. These include commands for buffer pointer movement, character deletion and insertion, pointer manipulations, and cassette input and output. See the PTP user's manual (Simons 1977:6-11) for further information. Another command which the user must know is the execute command, EXC. All of the programs are stored in buffer 1. In order to run a program, set the buffer pointer to buffer 1 (SBP command) and then use the MFL (move forward a line) command to step through the programs in the program buffer. When the desired program is found (the buffer pointer immediately precedes the name of the program) give the OEXC command to run the program.

## 2.1 Input

### 2.1.1 Enter a gloss file

Position the buffer pointer to the beginning of buffer 2. Use the INS (insert) command to enter the gloss file. Each entry in the file must begin with the double cross, #, character and end with a CRLF (carriage return/line feed). The first entry is a mnemonic identifier for the list. The remaining entries are the glosses for the word list items. They should begin with a number for easy reference. A sample gloss file is the following:

```
#ENG
#1. head
#2. hair
#3. nose
#4. eye
etc.
```

When the gloss file is fully entered, enter a header block and save the file on tape (see PTP user's manual, section 3.6). Delete the header block before proceeding. Next run "copy 2 to 3" (4.1.1) to copy the gloss file into buffer 3.

### 2.1.2 Enter the first word list

The gloss file now copied into buffer 3 may be thought of as a blank word list. The user's task is now to fill in the word list. All words are entered directly in front of the corresponding gloss. Position the buffer pointer to the beginning of buffer 3 and enter the mnemonic identifier for the word list. It must be three characters in length. To enter each word, position the buffer pointer to the beginning of the gloss and then enter the appropriate

word using the INS command. At the end of the word, press ESC. (escape) to terminate the entry. No CRLF is entered; thus the word and the gloss run together on the same line. Give the MFL command to advance to the next gloss. Digraphs may not occur in the orthography used for entering the word lists. The phonemic analysis, phonostatistic analysis, and comparative method programs require that all phones be encoded by a single character. Therefore, a suitable computer orthography must be worked out ahead of time (see PTP user's manual, section 2, for a discussion of the character set).

Before entering the word lists, the lists should be compared item by item in order to make cognate decisions. The decisions are indicated by assigning words to cognate sets (Carroll and Dyen 1962; Sanders 1977:36-7). Two words assigned to the same cognate set are judged to be cognate; words in different cognate sets are judged to be non-cognate. The first character in every word list entry is its cognate set identifier. In general, all cognate set identifiers in the first list entered will be the number one. As additional cognate sets are needed, they are assigned in numerical order. After nine is reached, the upper case letters are used beginning with A. The completed word list should appear as follows:

```
BIL#ENG
ltabana#1. head
lkataa#2. hair
lnora#3. nose
lmata#4. eye
etc.
```

### 2.1.3 Extract the word list alone

Run "extract list" (4.1.2) to copy the words without glosses into buffer 4. The word list in buffer 4 should now appear as follows:

```
BIL
ltabana
lkataa
lnora
lmata
etc.
```

Enter a header block for the word list and save the word list on tape.

## 2.1.4 Phonemic analysis

If the phonemic analysis programs are to be run on the current word list, it is best done at this stage while the list alone is standing in buffer 4. Delete the header block for the word list and then proceed with the phonemic analysis programs as outlined in section 2.2. Continue with 2.1.5 when phonemic analysis is completed.

## 2.1.5 Entering additional word lists

For entering additional word lists there are three options:

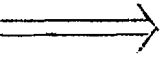
- (1) Edit the previous word list. Go to 2.1.6.
- (2) Edit a former word list which was saved on tape. Go to 2.1.7.
- (3) Type the whole list into a clean buffer. Go to 2.1.8.

If all the lists have been entered and saved on tape, go to 2.1.9.

## 2.1.6 Edit the previous word list

The word list combined with glosses that remains in buffer 3 is edited to represent the new list which is now being entered. If words in the new list are identical to words in the previous list, leave them untouched. If words in the new list are cognate but have a different pronunciation, edit them to represent the proper form in the new list by using the PTP immediate mode text editing commands. If words in the new list are non-cognate, delete the previous word and enter the new word with a different cognate set identifier. In using this approach it is best to enter the lists in such an order that the most similar lists are dealt with in succession. In this way the amount of work involved in entering the data is minimized. When editing is complete, go to 2.1.3.

The following example is given to illustrate what is meant by editing the word list. The list on the left is the original list; the list on the right is the edited second list. In the original list, those characters which were deleted are underlined. In the second list, those characters which were inserted are underlined.

BIL#ENG ltabana#1. head lkataa#2. hair lnora#3. nose lmata#4. eye etc.		BL2#ENG ltabana#1. head lgataa#2. hair <u>2usu</u> #3. nose lmata#4. eye etc.
---	---	--

### 2.1.7 Edit a word list saved on tape

First clear buffer 3 by running "zero 3" (4.1.3). Then read the desired list into buffer 3. Run "merge glosses into list" (4.1.4) to make a combined gloss and word list in buffer 3. Edit the word list as in step 2.1.6 above. Go to 2.1.3.

### 2.1.8 Enter the whole list into a clean buffer

First run "zero 3" (4.1.3) to clear buffer 3. Then run "copy 2 to 3" (4.1.1) to put the gloss file into buffer 3. First enter the three character mnemonic identifier and then all the words with their cognate identifiers. Follow the procedure used in step 2.1.2. Go to 2.1.3.

### 2.1.9 Merge word lists into a single master file

Before any further analysis can be done, the individual word lists that have been saved on tape must be merged into a single master file. The user must first select the set of word lists which he wishes to analyze. Then run "merge lists" (4.1.5) to perform the merge. The merged master file will be created in buffer 2.

The merge program is interactive and will merge any number of lists from two to ninety-nine. When the program is run, the message "glosses" will appear on the screen. This is the user's prompt message that the program is in the view mode and ready to read in the gloss file for the word lists. Mount the proper tape, position it before the header block by means of the digital counter on the tape drive, and then put the drive into play mode. All characters read from the tape will be echoed onto the screen but not inserted into the buffer. When the header block appears, press the ESC key. The program then executes a cassette read command and the gloss file is read into buffer 2. When the reading is complete the prompt question "more or no?" appears. At this point turn off the tape drive and ready the next word list for reading by mounting and positioning the proper tape. PTP is now in the immediate mode with the buffer pointer preceding the question "more or no?". Answer this question of whether or not there are more lists to merge by positioning the buffer pointer in front of the word which is the answer--"more" if there are more lists, "no" if the merging is completed. Then type # to return PTP to program control. If the response was "more", the prompt message "view" appears. This indicates that the program is executing a cassette view routine in anticipation of reading in a word list. Read in the word list as described above for the

gloss file. When the reading is complete, the word list will be merged into the master file in buffer 2. When this begins to happen (it will be obvious in watching the display--echoing of characters as they are read in stops, and sequencing through the word list begins) turn off the tape drive and ready the next word list. Answer the question when it appears. If the response is "no" then the program terminates with the buffer pointer at the beginning of the merged master file.

For the three word lists BIL, BL2, and GAL with the ENG gloss file, the merged master file would appear as follows:

```
#ENG
BIL
BL2
GAL
#1. head
ltabana
ltabana
ltabana
#2. hair
lkataa
lgataa
2kona
#3. nose
lnora
2usu
2uyu
#4. eye
lmata
lmata
lmata
etc.
```

If the master file is to be saved on tape, type a header block. After saving the file, delete the header block. Go to 2.3 to begin lexicostatistic analysis of the merged word lists.

## 2.2 Phonemic analysis

The programs in the phonemic analysis set aid the user in determining the phonemes of a language and their allophones, in determining the distribution of phonemes, and in determining the structure of syllables and words. With the results of the analysis stored in a buffer, it would be possible to convert automatically all phonetic transcriptions into phonemic ones. However, phonetic data are required for the phonostatistics programs in 2.5 and 2.7.

The comparative method in 2.6 must determine regular correspondences between phones in order for the refined phonostatistics program in 2.7 to work properly. Thus the phonetic word lists will not be converted into phonemic ones. Nevertheless, the comparative method is generally applied at the phonemic rather than the phonetic level. In this case, since the comparative method programs in 2.6 will actually be applied to phonetic word lists, it is necessary to combine the results of phonemic analysis with the results of the comparative method in order to determine the regular correspondences between dialects at the phonemic level. Comparing the results of the phonemic analysis of different word lists may show differences in the allophonic manifestations of phonemes, in the distribution of phonemes, or in the structure of words and syllables. If such differences occur, additional phonological isoglosses can be posited (see section 2.6.3).

A word list may not contain enough data to perform a complete phonemic analysis. For the sake of improving the phonemic analysis, the user may add words to the list in buffer 3. It will be necessary to follow the same format exactly--a fake cognate set identifier, the language word, a #, the gloss, a CRLF--for each added entry. Run "extract list" (4.1.2) to copy all the words without their glosses into buffer 4. If the user has more extensive data on only one or two dialects, the complete analysis can be done on those. Then the results obtained from any less complete analyses can be interpreted in the light of the more complete analysis of a related dialect.

### 2.2.1 Preparation

Before performing the phonemic analysis, it is necessary to strip the mnemonic identifier and the cognate set identifiers from the word list in buffer 4. To do this, run "words alone" (4.2.1). The result would appear as follows:

```
tabana
kataa
nora
mata
etc.
```

### 2.2.2 Phone occurrences

Run "phone occurrences" (4.2.2) to compile in buffer 5 a list of all the phones which occur in the word list and their frequencies of occurrence. The # item in the list tells how many words occur in the list. The output is an alphabetized list of the following format:

```
# 285
a 132
b 024
d 028
e 094
etc.
```

That is, the list contains 285 words, a occurs 132 times, b occurs 24 times, and so on.

If any of the phones listed looks suspicious--as though it may be an error in hearing, transcribing, or typing--run "find examples" (4.2.6) to find all examples of the suspicious phone. Before running the program, use the immediate mode to replace the double hyphen in the program with the character which is to be searched for. When the program terminates, every word containing that phone will be listed with its gloss in buffer 8. If this investigation does lead to discovery of an error, make the change in the word list.

In preparation for the "phone co-occurrences" program, copy the list of phones which occur onto a blank phone co-occurrence matrix as the row and column headings.

### 2.2.3 Phone co-occurrences

Run "phone co-occurrences" (4.2.3) to compile in buffer 7 a list of all the phone co-occurrences which occur in the word list and their frequencies of occurrence. In the output table, the double cross character, #, is used to represent a word boundary. Thus, #-a represents word initial a and a-# represents word final a. The output is an alphabetized list of the following format:

```
a-# 063
a-b 006
a-d 009
a-e 017
etc.
```

That is, a is word final 63 times, a is followed by b 6 times, and so on.

The table of results should be copied into the blank co-occurrence matrix prepared in 2.2.2. The first phone is the index to the rows of the matrix, the second phone is the index to the columns of the matrix, and the number of occurrences should be copied into the cell at the intersection of the proper row and column. By examining this matrix the user may discover the complementary distribution of allophones of a single phoneme. First compare the matrix rows of phonetically similar phones. If a pair of rows shows complementary distribution (that is, where one row has a filled-in cell, the other row has a blank one, and vice versa), those two phones are variants of a single phoneme which are conditioned by the following segment. Next compare the matrix columns of phonetically similar phones. If a pair of columns shows complementary distribution then the two phones are variants of a single phoneme which are conditioned by the preceding segment. The co-occurrence matrix also provides information on the consonant clusters which occur, the vowel clusters which occur, and transition frequencies.

If any sequence looks suspicious (for example, there is only a single example which refutes a complementarity hypothesis or a hypothesis as to the possible clusters that can occur) run "find examples" (4.2.6) to find all examples of the suspicious sequence. The program is also used to extract a list of supporting evidence for including in a write-up of the results. Before running the program, use the immediate mode to replace the double hyphen in the program with the two character sequence which is to be searched for. If word boundary is one of the characters, it must be entered as CRLF, not as #. When the program terminates, every word containing that sequence will be listed with its gloss in buffer 8. If in checking out the examples any errors are found, they should be corrected in the word list. If enough errors are found, it may be of advantage to go back to 2.2.2.

#### 2.2.4 Word shapes

The next program translates every word in buffer 4 into a sequence of C's and V's (and any other appropriate symbols) in order to study the internal structure of words. In addition, the words are syllabified. Before the program can operate, the user must enter the table for translating segments and the table of syllabification rules.

The table of phone occurrences in buffer 5 is used as the foundation for the translation table. Following the first character in each line of the table, insert the character into which the first character should be translated. For the first run translate all consonants into C and all vowels into V. Special characters should be translated to themselves--for instance, a stress mark should remain a stress mark. Only word boundaries, #, may



translate into word boundaries. If additional word boundaries are created, the "find examples" program will not work properly. A sample translation table would appear as follows:

```
## 285
aV 132
bC 024
dC 028
eV 094
etc.
```

The table of syllabification rules is entered into buffer 6. First determine the rules that correctly place syllable breaks in the words of the language. For instance, for the Biliau language of Papua New Guinea the following three rules were used (Simons and Simons 1977:5):

```
VCV → V.CV
VCC → VC.C
VV → V.V
```

The above rules are not ordered. If the rules for a language must be ordered, be sure to put the first rule at the top, and so on. To enter a rule into the table of syllabification rules, first count the number of characters in the left part of the rule and enter the number. Next enter the left part of the rule. Then count the number of characters preceding the syllable break (the period) in the right part of the rule and enter the number. Finally, end the rule with a CRLF. The proper form of the above three rules is the following:

```
3VCV1
3VCC2
2VV1
```

Now run "word shapes" (4.2.4) to compile in buffer 7 a table of all the word shapes that occur and their frequency of occurrence. The output is an alphabetized list of the following format:

```
CV 023
CV.CV 135
CV.CVC 043
CVC 054
etc.
```

That is, there are 23 words that are simply CV, 135 that are CV.CV, and so on.

The output is used to study the distribution of syllable types within the word and to define the possible structure of a word with respect to the syllable types and where they can occur. If stress is marked in the transcriptions, the above output will allow the user to study the distribution of stress with respect to both syllable types and position in the word.

By rerunning this program (and the next) it is possible to check out hypotheses about restrictions on the distribution of phonemes and about allophonic conditioning factors that are related to position in syllable, position in word, or placement of stress. Before rerunning "word shapes" it is necessary to first run "extract list" and then "words alone". If it is suspected that nasals, for instance, are the only consonants that can close a syllable, change the translation table in buffer 5 to translate the nasals into some unique character such as N. Rerun the program. If the hypothesis is true, all closed syllables will end with N; none will end with C. If it is suspected that the quality of vowels is conditioned by the placement of stress, translate the vowel phones thought to occur in unstressed syllables to U. Rerun the program to see if the distribution of U and V is indeed conditioned by stress placement. If the distribution of a particular phone is of special interest, translate it to itself and rerun the program.

To find all the words which exemplify a certain word shape, either to compile a list of supporting evidence or to check out counter-examples to hypotheses, run "find examples" (4.2.6). Before running the program, use the immediate mode to replace the double hyphen in the program with the word shape which is to be searched for. One can insert a whole word shape or part of one. If a whole word shape is to be searched for, CRLF must be inserted as the first and the last character of the search string. When the program terminates, every word with that shape will be listed with its gloss in buffer 8.

## 2.2.5 Syllable shapes

Run "syllable shapes" (4.2.5) to compile in buffer 7 an alphabetized list of all the syllable types that occur along with a count of their frequency of occurrence. The double cross character, #, is used to denote word boundaries. By using this symbol, the table of results distinguishes between initial syllables (eg. #CV), medial syllables (eg. CV), and final syllables (eg. CV#). Syllables that are both initial and final contain two #'s (eg. #CV#, a one syllable word). The table of

results has the following format:

```
#CV 242
#V 043
CV 052
CV# 285
```

That is, there are 242 words which begin with a CV syllable, 43 words begin with a V syllable, 52 words have a medial CV syllable, and 285 words end with a CV syllable.

As discussed in the previous section, this program is used with "word shapes" to test hypotheses about phoneme distribution and allophonic conditioning. The "find examples" program is used to compile in buffer 8 a list of examples of any syllable type. Before running the program, use the immediate mode to replace the double hyphen in the program with the syllable type which is to be searched for. If an initial or final syllable is to be searched for, CRLF must be entered in place of #. Any medial syllable border must be marked by the period which is the symbol for a medial syllable break. Thus, to find examples of medial CV syllables, replace the double hyphen with ".CV." .

## 2.3 Lexicostatistic analysis

### 2.3.1 Count cognates

Run "count cognates" (4.3.1) to count the number of cognates between every possible pairing of word lists in the master list. The results are tabulated in buffer 4. The results are in the following format:

```
092 BIL-BL2
075 BIL-GAL
054 BIL-GED
etc.
```

The first entry in the table of results states that there are 92 cognate forms between the BIL list and the BL2 list. The count of cognates is the absolute number of cognates. An electronic calculator can be used to convert the numbers in the display to percentages. Divide the number in the display by the number of items in the word list and multiply by 100.

### 2.3.2 Build a matrix of cognate counts

Run "build cognate matrix" (4.3.2) to convert the list of cognate counts in buffer 4 into a matrix in buffer 6. The matrix is square. The mnemonic identifiers are in the first row and column and the diagonal is filled with asterisks. That is,

```
*** BIL BL2 GAL GED
BIL *** 092 075 054
BL2 092 *** 074 053
GAL 075 074 *** 056
GED 054 053 056 ***
```

### 2.3.3 Permute the matrix of cognate counts

Permuting the matrix of cognate counts (that is, changing the order of the rows and columns) allows the user to see relations between the dialects that might otherwise remain obscured if the matrix were left in its original order. For a discussion of matrix permutation in relation to recognizing patterns of divergence and convergence in a lexicostatistic matrix, see Simons (1977b). To permute the matrix of cognate counts, run "permute matrix" (4.3.3). This is an interactive program. The program begins by entering the immediate mode with the buffer pointer at the beginning of the top row. Set the Aux pointer (SAX command) to the beginning of the mnemonic identifier of the column which is to be moved. That is, MFC to the proper character and then SAX. Then set the buffer pointer to precede the mnemonic identifier of the column in front of which the other column is to be moved. Give the # command to return to program control; the proper column and its corresponding row will be moved to the desired position. After the move is completed the program displays the result and returns to the immediate mode. Repeat the steps outlined above to move another column and row. To terminate the program, type # without moving the buffer pointer from its position in the upper left hand corner of the matrix.

## 2.4 Lexical isogloss analysis

### 2.4.1 Extract a lexical isogloss file

A lexical isogloss file contains only the cognate set identifiers and the glosses from the merged master file. To extract such a file, first insert a unique one character identifier for each word list in front of the list's mnemonic identifier. For example,

```
#ENG
aBIL
bBL2
cGAL
etc.
```

Then run "lexical isoglosses" (4.4.1) to extract a lexical isogloss file into buffer 5 of the merged master file in buffer 2. The lexical isogloss file for the merged master file shown in 2.1.9 is as follows:

```
abc#ENG
111#1. head
112#2. hair
122#3. nose
111#4. eye
etc.
```

If the isogloss file is to be saved on tape, type a header block, save the file, and delete the header block.

### 2.4.2 Analysis of lexical isoglosses

The program "isogloss analysis" (4.4.2) aids the user's study of lexical isoglosses in two ways. First, the program sorts the file of lexical isoglosses. This puts all the word list items with the same geographical distribution of cognates together in the isogloss file. The original isogloss file shown on the left, when sorted will appear as on the right.

ORIGINAL	SORTED
abcdefg#ENG	abcdefg#ENG
1111111#1. head	1111111#1. head
1121222#2. hair	1111111#3. nose
1111111#3. nose	1111111#5. mouth
1111222#4. eye	1111222#4. eye
1111111#5. mouth	1111222#7. tooth
1122222#6. ear	1121222#2. hair
1111222#7. tooth	1122222#6. ear

From the sorted isogloss file we see that three items #1, #3, and #5 do not differentiate the dialects and there is no isogloss. Items #4 and #7 are examples of an isogloss distinguishing dialects a through d from dialects e through g. Item #2 indicates an isogloss which distinguishes dialects a, b, and d from the other four dialects. Item #6 indicates an isogloss separating a and b from the remaining dialects. From the sorted isogloss file, the isoglosses can be drawn directly onto a map and each isogloss labelled with the numbers of the word list items that evidence it.

The second way in which the program can aid the analysis of isoglosses is by allowing the user to change the order of the word lists in the isogloss file. For instance, by swapping the order of dialect c and dialect d in the preceding example, the isogloss pattern for #2 would become "1112222#2. hair", rather than the original "1121222#2. hair". (The other six items remain unchanged since c and d are cognate in all other items.) Such a change, by putting all the like elements in an isogloss pattern together, makes it easier for the analyst to see the various isogloss patterns that occur. In general, the best ordering of the word lists puts the most closely related dialects in adjacent spots. The best ordering can first be found by permuting the matrix of cognate counts as in 2.3.3. In most cases the best ordering will also be a geographical ordering.

To perform the analysis run "isogloss analysis" (4.4.2). The program first allows the user to reorder the word lists, then it sorts the file. The reordering phase of the program is interactive. The user achieves the desired order by moving lists one at a time to the leftmost position. Any ordering of the lists is achieved by moving the lists in the proper sequential order. The program begins by going to the immediate mode with the buffer pointer preceding the first character of the first line--the line of single character identifiers for the lists. Use MFC to position the buffer pointer in front of the identifier for the list which is to be moved to the first position. Then type #. Execution returns to program control and the column will be transferred to the initial position and deleted from its original position. The result is displayed and the program again goes to the immediate mode. Repeat the steps given above to move more lists. When the isogloss patterns are in the desired order, type # without moving the buffer pointer from the front of the first line. The program will then execute the sort routine, display the sorted file, and terminate. Rurun the program to explore other orderings of the word lists. If the user wishes to study the isogloss patterns of only a subset of the word lists, move those word lists to the leftmost positions in the isogloss file and ignore the other lists.

## 2.5 Phonostatistic analysis

### 2.5.1 Pre-program data formatting

Before the phonostatistic analysis may be made, the user must manually do some formatting of the merged master file in buffer 2. The phonostatistics program compares cognate words character by character and tabulates the degrees of difference between corresponding phones. The purpose of the data formatting is to ensure that the corresponding phones in cognate words occur in the same position in the word. This is done by inserting a slash where a phoneme has been lost through a process of sound change. Where one word is longer than another because it contains an additional morpheme, hyphens are inserted into the shorter word. Using both slash and hyphen makes it possible to distinguish between differences in form due to sound change and differences in morphology. After formatting, all forms for a given word list item that are cognate (that is, have the same cognate set identifier preceding them) must be of the same length. Non-cognate forms are not compared and so relative length is immaterial. A correct formatting example follows:

ORIGINAL	FORMATTED
#68. snake	#68. snake
ltofi	ltofi
ltoi	lto/i
2wata	2--wata
ltohi	ltohi
2waa	2--wa/a
2wawaa	2wawa/a
loi	l/o/i

To facilitate the insertion of slashes and hyphens, the user should insert the following simple programs at the beginning of buffers 1 and 2:

At the beginning of buffer 1:   INS/\$#

At the beginning of buffer 2:   INS-##

To insert a slash, position the buffer pointer to the point where the slash is to be inserted and then given the command EXC. To insert a hyphen, position the buffer pointer to the position where the hyphen is to be inserted and give the command 2EXC. Delete the INS/\$# and INS-## programs before proceeding to the next step.

### 2.5.2 Format checking

After the pre-program data formatting is completed, run "check format" (4.5.1) to confirm that all cognate forms are indeed of equal length. If any are not, the phonostatistics program will not work properly. As the "check format" program runs, the display is constantly changing as it steps through all the pairs of words. If the program finds a pair of cognate words that are not of the same length, the program will terminate and return to the immediate mode. The display will show the buffer pointer following the message "\*\*\*ERROR\*\*\*" which is inserted into the word on which the length error occurred. The Aux pointer is at the end of the other word being compared. Press TRA to find the other word. Use the immediate mode editing commands to remove the "\*\*\*ERROR\*\*\*" message and do the proper formatting to equalize the lengths. Be sure to check other words in the same cognate set, for it is likely that other cognate words may be involved. After editing is completed, run the program again to check for more formatting errors. If the program terminates without an "\*\*\*ERROR\*\*\*" message, the file is properly formatted. Type a header block and save the formatted file on tape. It will be needed again in 2.7. Delete the header block after the file is saved.

### 2.5.3 Enter the degrees of difference table

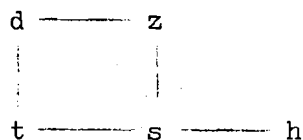
The approach to phonostatistics used in the Word List Analysis Package is essentially the method which was devised by Joseph Grimes (Grimes and Agard 1959, Grimes 1964) and extended to application on comparative word lists by Howard McKaughan (1964). In this method, corresponding phones in cognate words are compared and scored quantitatively as to the phonetic "degree of difference" between them. Identical phones differ by zero degrees. Phones differing by a minimal feature difference in only one phonetic dimension differ by one degree of difference, and so on. Grimes and McKaughan specify each of the phones in their corpus by numerically encoding their phonetic "rank of stricture" in each of a number of phonetic dimensions. The degree of difference between corresponding phones can then be automatically computed by comparing the phonetic specifications of the two phones. Ladefoged (1968) used a similar approach in which phones were specified by generative distinctive features.

In a review of phonostatistic methods (Simons 1977a:177-9), I have suggested that the best method of assigning degrees of difference to corresponding phones is for the linguist to apply his experience with the languages being compared, and with



language in general, to assign degree of difference in terms of minimal steps of sound change. In many cases the rank of stricture and distinctive feature approaches yield similar results, but in some cases they do not. For instance, a change from bilabial point of articulation to velar point of articulation (which can be a natural and minimal sound change where labiovelar consonants are found) is handled very naturally by the distinctive feature approach as a change in the "anterior" feature. However, it represents a high degree of difference in the rank of stricture approach where alveolar and alveopalatal points of articulation must be counted as intervening. Conversely, the rank of stricture approach very naturally handles the change from voiced dental fricative /ð/ to semivowel /y/ to high vowel /i/ as minimal steps in the dimension of constriction of the air stream, while in the distinctive feature approach these changes require a messy reshuffling of many features--consonantal, continuant, vocalic, anterior, coronal, high, front. Furthermore, there are some common changes that neither approach can handle as a natural and minimal change, for instance, /s/ to /h/ or /t/ to glottal stop. (See Simons 1977a:177-9 for a more complete discussion.)

In order to achieve the most natural assignment of degrees of difference to sound correspondences, I suggest that the investigator compute degrees of difference in terms of minimal steps in sound change. To do this, he must first compile a list of all the sound correspondences in the corpus of data. He must then examine the correspondences and posit the network of minimal steps which explain sound change in the corpus of data. For instance, given the correspondences, t:s, s:h, t:h, d:t, d:z, d:s, and z:s, one would construct the following network of minimal steps:



The degree of difference between two phones is then the number of arcs in the network that are traversed in going from the first phone to the second via the shortest path. From the above network, sample degrees of difference are:

h:h 0	t:t 0	d:d 0
s:s 0	t:s 1	d:t 1
s:h 1	t:h 2	d:z 1
		d:s 2
		d:h 3

The program "check format" which has already been run in the previous section, compiles a list of all the correspondences that occur in the merged master file. It does this at the same time it is checking the format. After the program terminates successfully, set the buffer pointer to buffer 3 to find the list of correspondences. Each item in the list is only two characters, the two corresponding characters, followed by a CRLF. The list is in random order. In this package of programs, degree of difference is assumed to be symmetrical. That is, the degree of difference for d:t equals the degree of difference for t:d. Thus all correspondences are given only once in a normalized form--the second character will always be lexically equal to or greater than the first. A sample list is as follows:

```
tt
aa
hs
st
ae
ht
dd
dt
etc.
```

From this list determine the network of minimal steps. Then go through the list item by item and insert the degree of difference measure for each correspondence following the second character. That is,

```
tt0
aa0
hs1
st1
ae1
ht2
dd0
dt1
etc.
```

The format here is a general one. It does not require that a minimal steps approach be taken. If the user prefers a rank of stricture or distinctive feature approach, degree of difference can be computed by those methods and then inserted into the list as above.

When the degrees of difference table is completely filled in, type a header block and save the table on tape. It will be needed again in 2.7. Delete the header block after the file is saved.

#### 2.5.4 Phonostatistics

Run "phonostatistics" (4.5.2) to perform a phonostatic analysis of the formatted master file in buffer 2. Each pair of lists is compared in turn. After each pair is compared, the program goes to immediate mode with the results in buffer 4. After the results for that pair of lists are written down, give the # command to return to program control and compute the statistics for the next pair. The results appear in a table of the following format:

```
BIL-GAL
w 120
c 508
d 308
0 286
1 160
2 042
3 016
4 004
5 000
```

The first line, BIL-GAL, tells that the following table is the result of the phonostatic comparison of the BIL list with the GAL list. The second line, w 120, tells that 120 cognate words were compared. The third line, c 508, tells that 508 corresponding phones were compared. The fourth line, d 308, tells that the total degree of difference over the 508 correspondences was 308. The remaining lines give a breakdown of the correspondences compared and tell how many correspondences had 0 degrees of difference, 1 degree of difference, and so on. Thus the sum of lines 0 through 5 equals the figure for c, and the figure for d equals the sum of the products of degrees of difference and the number of correspondences with that degree of difference. That is,  $508 = 286 + 160 + 42 + 16 + 4 + 0$  and  $308 = (0 \times 286) + (1 \times 160) + (2 \times 42) + (3 \times 16) + (4 \times 4) + (5 \times 0)$ .

An electronic calculator can be used to convert these statistics into more meaningful numbers for the sake of comparison. Dividing c by w gives the average number of correspondences compared per word--the average length of a word. Dividing d by c gives the average degree of difference for corresponding phones. Dividing d by w gives the average number of phonetic features that are different (eg. degrees of difference) between cognate words. The 0 through 5 lines can be made into more useful statistics by converting them to percentages. This is done by dividing each by c and multiplying by 100. When these are converted to percentages, the distributions of degrees of difference between various pairs

of lists can be compared. These computed statistics for the above table are:

BIL-GAL

c/w = 4.23

d/c = 0.61

d/w = 2.57

Distribution of c with respect to d:

0	56%
1	31%
2	8%
3	3%
4	1%
5	0%

The d/c measure between any two pairs is directly comparable. In order to use d/w to compare all the relations between lists, it would be best to normalize all d/w scores to an average length of word. If there is a variation in c/w (correspondences per word), then variation in d/w will be related to variation in c/w as well as to variation in the actual average degree of phonetic difference between the lists. To control for the variation in c/w, first compute the average c/w. Then compute the normalized d/w by multiplying the corresponding d/c by the average c/w. A simpler alternative would be to compute all d/w measures on the basis of a word of modal (rather than mean) length. That is, if four character words are the most frequent in the languages, then compute d/w on the basis of four segment words. That is,  $d/w = d/c \times 4$ .

To make phonostatistic measures more comparable to lexico-statistic measures, a phonostatistic measure can be converted to a percentage. To do this it is necessary to determine the maximum degree of difference possible for the given data. The percentage of phonostatistic difference would be the percentage of observed degrees of difference out of the maximum possible degrees of difference. The percentage of phonostatistic sameness is then 100 minus the percentage of phonostatistic difference. That is, percentage of phonostatistic sameness (PS) would be defined by:

$$PS = 100 - ((d \times 100)/(c \times \max))$$

For the above data, if max is defined as 5 degrees of difference, the percentage of phonostatistic sameness is:

$$100 - ((308 \times 100)/(508 \times 5)) \text{ or } 88\%$$

That is, the two lists are 88% phonetically similar in that only 12% of the phonetic features that could possibly be different are in fact different. It must be remembered that this is the degree of similarity between cognate words only.

## 2.6 Comparative method

### 2.6.1 Pre-program data formatting

Before the comparative analysis can be made, further manual formatting of the data must be done. One must begin with the formatted merged master file produced in 2.5.1. The formatting in 2.5.1 was performed to ensure that all cognate words were of the same length and that corresponding phones were in the same position in the word. For the comparative method there is an additional requirement that all the words listed under a gloss must be of a single cognate set. There are four cases to be dealt with, each of which is discussed in a paragraph below: (1) all the words listed under a gloss are cognate, (2) none of the words listed under a gloss are cognate and there is nothing to compare, (3) the majority of the words under a gloss are in one cognate set with a few odd non-cognate forms, and (4) more than one cognate set is widely represented under a single gloss and thus there is more than one set of forms to compare.

(1) Where all the words listed under a gloss are cognate, run "prepare a set" (4.6.1) to delete the cognate set identifiers from before the words. Position the buffer pointer to the beginning of the gloss (that is, preceding the #) and then set the Aux pointer (SAX command). Find the "prepare a set" program in the program buffer and run it. All cognate set identifiers in that set of words will be deleted as in the following example:

#31. mountain		#31. mountain
ltolo		tolo
ltolo		tolo
ltolu	→	tolu
ltolo	→	tolo
ltolo		tolo
ltolo		tolo
#32. sand		#32. sand
etc.		etc.

(2) Where the words listed under a gloss are not cognate and there is no data to compare, run "delete a set" (4.6.2) to delete the gloss and the complete set of words. Position the buffer pointer to the beginning of the gloss (that is, preceding the #) and then set the Aux pointer (SAX command). Find "delete a set" in the program buffer and run it. The result will be that all lines from the Aux to the next # will be deleted.

(3) Where the majority of words under a gloss are in one cognate set with a few odd non-cognate forms, run "prepare a set" (4.6.1) to delete all the odd forms so that only cognate forms remain in the set. First position the buffer pointer to the beginning of the gloss (that is, preceding the #). Then insert the number of the cognate set that is to be preserved. Position the buffer pointer to precede the number and set the Aux pointer (SAX command). Find "prepare a set" in the program buffer and run it. The result will be that any form not in the specified cognate set will be replaced by a string of hyphens of the same length as all forms in the cognate set. The string of hyphens means that no comparative data is available for that word. The cognate set identifiers are also deleted. The number preceding the gloss is preserved, however. This serves to identify the following data set as to its cognate set identifier. For instance,

1#32. sand		1#32. sand
lhone		hone
1/one		/one
2pususu	→	----
1/one	→	/one
3balo		----
lfone		fone
#33. water		#33. water
etc.		etc.

(4) Where more than one cognate set is widely represented under a single gloss, run "copy a set" (4.6.3) to make a complete copy of the gloss and the words. Position the buffer pointer to the beginning of the gloss (that is, preceding the #) and then set the Aux pointer (SAX command). Find "copy a set" in the program buffer and run it. The result will be that the whole set of lines from the Aux to the next # will be copied and inserted at the end of the current set. Then run "prepare a set" as outlined in (3) above to prepare the different sets to exemplify different cognate sets. For instance,

#33. water		#33. water		1#33. water
lkaho		lkaho		kaho
lka/o		lka/o		ka/o
2tai	→	2tai	→	----
2tai	→	2tai	→	----
2tei		2tei		----
lkafo		lkafo		kafo
#34. fire		#33. water		2#33. water
etc.		lkaho		---
		lka/o		---
		2tai	→	tai
		2tai	→	tai
		2tei		tei
		lkafo		---
		#34. fire		#34. fire
		etc.		etc.

### 2.6.2 Count correspondence sets

Run "count correspondence sets" (4.6.4) to compile a list of all the correspondence sets that occur in the formatted master file and count the number of times each correspondence set occurs. Before running the program, first insert a unique one character identifier for each word list in front of the list's mnemonic identifier, as illustrated previously in 2.4.1. Then run the program. The results will appear in buffer 5 as a list of correspondence sets with their frequency of occurrence. The first entry in the list is the unique one character identifiers for the word lists, followed by htd (hundreds, tens, digits) which label the columns of the frequency count. For example,

```
abcde htd
fffff 006
ooooo 054
nnnnn 013
aaaae 008
etc.
```

That is, in six cases language d has /h/ where all the other languages have /f/, and so on. The correspondence sets are listed in an unsorted order.

### 2.6.3 Phonological isogloss analysis

Under the heading of phonological isogloss analysis come all the steps taken to analyze the list of occurring sound correspondences and determine the regular sound correspondences. Two programs are used: "isogloss analysis" (4.6.5) is used to reorder the columns in the list of correspondences and sort the list, and "find examples" (4.6.6) is used to find all the examples of a particular correspondence.

In general, the regular correspondences will be those which occur most frequently. In order to sort the list of correspondences into the order of frequency, run "isogloss analysis" as described already in 2.4.2. Four columns must be moved to the front in the following order: (1) the space column preceding the numbers, (2) the d (digits) column, (3) the t (tens) column, and (4) the h (hundreds) column. Then pass control to the sort routine.

Often it is helpful to sort all the correspondence sets with respect to a particular reference language, or group of languages. To do this, run "isogloss analysis" and move the desired language or languages to the front before sorting. The result is that all correspondences with the same phone in the reference language

will be listed together. Thus all correspondence sets with a in the reference language will be listed first, correspondence sets with b in the reference language come next, and so on. By comparing the frequency of occurrence of all correspondences involving a given reference language phone, one can determine the regular correspondence set involving that phoneme.

When more than one correspondence set for a single reference language phoneme has a high frequency of occurrence, one must look at all the examples of the correspondence sets to determine if each of the different correspondence sets occurs in a different environment. When writing up results, one needs to compile lists of supporting evidence. To extract examples of the correspondence sets, run "find examples". Before the program can be run, replace the double hyphen in the program listing with the correspondence set to be searched for. Thus, to find all examples of the correspondence set a:a:a:e:e, the `pi--$` in the program is changed to `piaaaee$`. The characters in the correspondence set must follow the order in which they appeared in the original results from the "count correspondence sets" program. Thus if the original order of one character identifiers is abcde, then all correspondence sets in the "find examples" program must be entered in the order of phone from language a first, then phone from b, then c, then d, and then e. Even if the "isogloss analysis" program has been used to reorder the languages in the correspondence sets, the original order of the correspondence sets must be used in the "find examples" program. The result of the "find examples" program is a list in buffer 7 of the items which contain the given correspondence set. This list contains only the number and gloss of the item, not the actual data. The data can be found by referring to a typed listing of the merged master file. In the case where an item originally had more than one cognate set, the cognate set identifier will precede the gloss of the item. For instance,

```
1#15. hand
2#23. blood
#32. sand
1#61. two
#85. give
```

In this instance, five examples of the correspondence were found. They occur in cognate set 1 of item #15 'hand', cognate set 2 of item #23 'blood', item #32 'sand', and so on.

The correspondence sets will include the hyphen character which means no comparative data were available for that particular dialect and word. (Remember that the loss of a phone is signalled by the slash.) Thus a correspondence containing a hyphen is in general an incomplete example of one of the other complete correspondence sets. The "isogloss analysis" program, when it



sorts the list of correspondence sets, will list the incomplete correspondence sets near the complete correspondence sets of which they are only incomplete examples. To simplify the results, the incomplete correspondence sets can be deleted and the frequency counts for the complete correspondence sets which they represent can be incremented accordingly.

When the regular correspondence sets have been determined, the irregular correspondence sets should be deleted from the list in order to prepare for the refined phonostatistics program. From the list of regular correspondence sets which remains in buffer 5, the phonological isoglosses can be extracted and drawn on a map. That is, any correspondence set in which not all members of the set are identical represents a phonological isogloss and isogloss lines can be drawn on a map to separate the points which have non-identical correspondence set members. If one wishes to label isogloss lines with the numbers of the items which exemplify the isogloss, run "find examples" to extract a list of these items. As previously discussed in 2.2, the results of phonemic analysis should be consulted at this stage to determine the correspondences at the phonemic level. Comparison of the phonemic analyses of different dialects may also yield differences in allophonic manifestations of phonemes, distribution of phonemes, or the structure of words and syllables. Such differences can be included in the isogloss map.

It may be desirable to make a second isogloss map showing the isoglosses defined by irregular correspondence sets. In this case the irregular correspondence sets can be saved into buffer 8 before deleting them from the main list in buffer 5. To do this, set the Save pointer to buffer 8 by executing 8SBP and then SSV. Before deleting a correspondence set from buffer 5 execute SVL to save it in buffer 8.

## 2.7 Refined phonostatistic analysis

After establishing the regular correspondence sets it is possible to perform a refined phonostatistic analysis in which the degree of phonological difference between cognate words is divided into two components--that portion of the difference which is due to regular sound correspondences and that portion which is due to irregular correspondences. The program requires three data files: (1) Buffer 2 must contain the merged master file formatted for phonostatistics which was saved on tape in 2.5.2. Buffer 2 must be cleared and the file read from tape. (2) Buffer 3 must contain the degrees of difference table produced in 2.5.3. (3) Buffer 5 must contain the list of regular correspondence sets. The

one character mnemonic identifiers in the first line of the file of regular correspondences must be in the same order as the mnemonic identifiers of the merged master file. If they are not, run "isogloss analysis" to put the correspondence sets into the proper order.

After the data files are in order, run "refined phonostatistics" (4.7). Each pair of word lists is compared in turn. After each pair is compared the program goes to immediate mode with the results in buffer 4. After the results for that pair of lists are written down, give the # command to return to program control and compute the statistics for the next pair. The results appear in a table of the following format:

```

BIL-GAL
w 104 016
c 488 020
d 274 034
0 286 000
1 148 012
2 038 004
3 014 002
4 002 002
5 000 000

```

The first column represents a tabulation of regular correspondences and the second column of irregular correspondences. The rows have the same meanings as in the "phonostatistics" program (see 2.5.4)-- words compared, correspondences compared, total degrees of difference, correspondences with 0 through 5 degrees of difference. The first row of the above table states that 104 of the words compared contained no irregular correspondences, while 16 word did contain at least one irregular correspondence. The sum of 104 and 16, that is 120, is the total number of words compared (see the table under 2.5.4). The second row tells that 488 of the correspondences were regular while 20 of them were irregular. The sum of 488 and 20, that is 508, is the total number of correspondences compared (see the table under 2.5.4). The third row tells that the 488 regular correspondences account for 274 degrees of difference, while the 20 irregular correspondences account for 34 degrees of difference. The sum of 274 and 34, that is 308, is the total degrees of difference (see table under 2.5.4). The remaining rows of the table are a breakdown of row c, giving the number of correspondences regular and irregular with 0 through 5 degrees of difference.

For the sake of comparison and interpretation, the figures in the above table are best converted to percentage scores. This is done by dividing each figure by the sum of the row (which equals the figure given by the "phonostatistics" program, see table under 2.5.4)

and multiplying by 100. That is,

BIL-GAL		
w	87	13
c	96	4
d	89	11
0	100	0
1	92	8
2	90	10
3	88	12
4	50	50
5	0	0

That is, 13% of the cognate words contain at least one irregular correspondence. While only 4% of the correspondences are irregular, they account for 11% of the phonological degrees of difference. This 11% of degrees of difference due to irregular correspondences says that 11% of the sound change observed relative to the two dialects does not conform to the classic Neogrammarian hypothesis of sound change without exceptions. Determining the source of this 11% discrepancy may prove significant for explaining language change within the area of the language survey. A possible source of discrepancy would be errors in the investigator's transcription of the word list. If this is not thought to account for all the discrepancy, two possible sources remain. (1) Borrowing may account for irregular correspondences. In this case, some of the forms counted as cognate are cognate only in a synchronic sense, but not in a diachronic one. (2) Sudden phonological innovations may account for the irregular correspondences. In this case the classic model of a constant, gradual, exceptionless sound drift is inadequate. There is something in the socio-cultural setting, such as a practice of word tabooing, which motivates sudden phonological change in individual lexical items (see for instance Keesing and Fifi'i 1969:167-168).

### 3. THE PTP PROGRAMMING LANGUAGE

The PTP programming language is described in "A User's Manual for PTP" (Simons 1977). Very briefly, PTP operates in two modes--an immediate mode and a program mode. In the immediate mode, standard text editing commands are executed and the results displayed immediately on the monitor screen. In the program mode, predicates and control structure are added to make a powerful programming language. In this mode text processing is done automatically according to a predefined program. Section 3.1 is a discussion of the changes which have been made to PTP since the user's manual was written. Section 3.2 gives a summary of PTP commands and special characters. This is provided as a reference guide for reading the programs in section 4.

### 3.1 Updates to the PTP user's manual

The following changes have been made: (1) addition of a TRS (TRade Save) command, (2) change of INC (INCrement) command and deletion of NUM (NUMber) command, (3) addition of indirect addressing of arguments, and (4) addition of zero arguments.

(1) Trade Save - The TRS (TRade Save) command has been added. It functions like TRA (TRade Aux) except that the buffer pointer and the save pointer are traded. TRS is assigned in the u key.

(2) Change in Increment - INC has been rewritten to accept an argument and to perform addition on a string of decimal digits rather than a single byte. The NUM command is thus no longer needed. The argument of INC can range from 0 to 255 and makes it possible to do addition. For instance, 54INC will add 54 to the current value of the number at the buffer pointer. The number is a string of decimal digits (eg. 0 through 9) and the buffer pointer must be positioned to precede the least significant digit. If a digit is 9 and it is incremented, it is replaced with 0 and the preceding digit is incremented. The buffer pointer is not affected; it remains on the least significant digit. Before initializing a number (which is done by inserting a string of digits), the programmer must determine what maximum value the number is likely to reach. Then he must initialize the number with enough digits to prevent an overflow. There is no limit to how many digits a number may contain.

(3) Indirect addressing of arguments - The k command loads the number at the Save pointer into the argument register. k takes an argument (1 to 3) which tells how many digits following the Save pointer are to be read as the argument. Thus the command 2kMFL will read the two characters following the Save pointer into the argument register and then move forward that many lines. The use of k is illustrated in the "phonostatistics" program (4.5.2).

(4) Zero arguments - A minimum argument of zero is now allowed for the four move commands (MFC, MFL, MBC, MBL), the four delete commands (DFC, DFL, DBC, DBL), and the increment (INC) command. When a zero argument is given for these commands, nothing happens. Therefore it is meaningless to type a zero argument in a program. The zero argument is encountered via the k command (indirect addressing of arguments) where it is useful in a variable argument for addressing the first row or column of a table, or for indicating a case where nothing is to be added to the count of results.

### 3.2 A summary of PTP commands and special characters

<u>n</u>	represents that the command may have a numeric argument, either digits or the <u>k</u> function. For moves, deletes, INC, and EXC, 0 is minimum value; for all others it is 1. Maximum is 3 for <u>k</u> , 16 for SBP and EXC, 255 for all others.
<u>  </u>	represents an argument string of any length
BP	an abbreviation for <u>B</u> uffer <u>P</u> ointer
nk	indirect addressing of argument--the n digits following the <u>S</u> ave pointer are read as the argument
nCPC	<u>C</u> o <u>P</u> y the n <u>C</u> haracters following <u>A</u> ux to the BP (BP updated)
nCPL	<u>C</u> o <u>P</u> y the n <u>L</u> ines following <u>A</u> ux to the BP (BP updated)
nDBC	<u>D</u> elete <u>B</u> ack n <u>C</u> haracters preceding BP
nDBL	<u>D</u> elete <u>B</u> ack n <u>L</u> ines preceding BP
nDFC	<u>D</u> elete <u>F</u> orward n <u>C</u> haracters following BP
nDFL	<u>D</u> elete <u>F</u> orward n <u>L</u> ines foll wing BP
nEXC	<u>E</u> X <u>e</u> <u>C</u> ute the program located in buffer n; if n equals zero, execute the program which begins at the BP
IMM	call the <u>I</u> MMediate mode processor from a program
nINC	<u>I</u> NCrement the number of which BP points to the least significant digit
INS, <u>  </u> ESC	<u>I</u> NSert the given argument string into the buffer at the BP in the immediate mode
INS, <u>  </u> \$	<u>I</u> NSert the given argument string into the buffer at the BP in program mode
nMBC	<u>M</u> ove the BP <u>B</u> ack n <u>C</u> haracters
nMBL	<u>M</u> ove the BP <u>B</u> ack n <u>L</u> ines
nMFC	<u>M</u> ove the BP <u>F</u> orward n <u>C</u> haracters
nMFL	<u>M</u> ove the BP <u>F</u> orward n <u>L</u> ines
MON	<u>M</u> ONitor the current buffer
SAX	<u>S</u> et the <u>A</u> u <u>X</u> iliary pointer to the BP

nSBP     Set the Buffer Pointer to the beginning of buffer n  
 SSV     Set the SaVe pointer to the BP  
 nSVC     SaVe n Characters following BP to the save pointer  
           (save updated)  
 nSVL     SaVe n Lines following BP to save pointer (save updated)  
 TRA     TRade the Aux pointer and the BP  
 TRS     TRade the Save pointer and the BP  
 cr     cassette read  
 cv     cassette view  
 cw     cassette write  
 pa     Does the aux pointer equal the BP?  
 npe     Are the n characters following the aux pointer equal to  
           the n characters following the BP?  
 npf     Are the n lines following the aux pointer equal to the  
           n lines following the BP?  
 pi,\_\_\_\_\$     is the string following BP equal to the argument string?  
 pl     Is the line following the aux pointer lexically greater  
           than the line following the BP?  
 pm     Is there more left in the current buffer?  
 nps,\_\_\_\_\$     search for the nth occurrence of the argument string  
 (     enter a procedure.  
 )     leave a procedure  
 :     jump to beginning of current procedure and execute again  
 ;     jump to end of current procedure, fall through right  
           parenthesis  
 ?     on true, fall through; on false, jump to semicolon  
 #     terminate the current EXC or IMM command  
 "     balanced double quotes delimit a comment  
 ✓     form feed for cassette write--fill to end of block with iota

#### 4. PROGRAM LISTINGS

In this section, complete program listings for all the programs in the Word List Analysis Package are given. In the introductory section for each set of programs, there is a guide to the buffer assignments for that set of programs. In each program listing there are two main parts --the listing and a structured program description. In the program listings, the spaces and CRLF's are strictly for the purpose of making the programs easier to read. In a text buffer, programs are stored in compact format with no spaces between commands and no CRLF's and indentations. Where a space is actually an essential part of the program (that is, it is a character in an argument string for INS or a predicate) it is represented in the program listing as an underline (   ). Where a CRLF occurs in an argument string, it is represented by a cent sign ( ¢ ). In the structured program description, the PTP control structure is copied from the program listing and filled in with prose statements which give the intent of the commands and predicates.

The program code for the entire Word List Analysis Package is 3½K in length. The total workspace available for both programs and data is 13K. In order to maximize the amount of workspace available for the data, the Word List Analysis Package is stored on tape in six load modules. These range in length from 325 bytes to 750 bytes. The user selects the module for the phase of analysis he is working on and reads in only those programs. This leaves over 12K of workspace available for data at all times. The organization of the modules and their lengths is as follows:

<u>Name</u>	<u>Sections for listings</u>	<u>Length in bytes</u>
Input	4.1	325
Phonemic analysis	4.2	750
Lexical analysis	4.3 and 4.4	700
Phonostatistics	4.5	600
Comparative method	4.6	600
Refined phonostatistics	4.7	<u>550</u>
TOTAL LENGTH		3525 bytes

#### 4.1 Input

The buffer assignments for the input programs are the following:

- Buffer 1 - Programs
- Buffer 2 - Gloss file; merged master file for "merge lists"
- Buffer 3 - Combined word list and gloss file; the list being merged in for "merge lists"
- Buffer 4 - Word list alone
- Buffer 5 - Prompt question "more or no?" for "merge lists"
- Buffer 6 - nnMFL# subroutine for "merge lists"

##### 4.1.1 Copy 2 to 3

```
"copy 2 to 3"
(3SBP SSV 2SBP
(pm? SVL MFL MON:;) 3SBP)#
```

##### 4.1.2 Extract list

```
"extract list"
(4SBP (pm? DFL:;) SSV
3SBP (pm? MON
(pi#? MFL MBC SVC MFC; SVC MFC:) :;)
4SBP)#

( (Clear buffer four) Set the Save pointer to buffer 4.
Go to buffer 3 to extract the word list from the combined list.
(More?
(Have we reached the gloss yet? Move to the next entry.
Save the CRLF at the end of the preceding entry;
No, save the character and advance to next:) :;)
Display results in buffer 4.)
```

##### 4.1.3 Zero 3

```
"zero 3"
(3SBP (pm? MON DFL:;))#
```



## 4.1.4 Merge glosses into list

```
"merge glosses into list"
(2SBP SAX 3SBP
  (pm? MON
    (pic$? DFC CPL TRA MFL TRA; MFC:);)
  3SBP)#

(Set Aux to the gloss file. Set BP to the word list.
  (More in word list?
    (End of the word? Yes, delete CRLF, copy gloss, and
      advance to next gloss in gloss file; No, next character:)
    ;;) Display result in buffer 3.)
```

## 4.1.5 Merge lists

```
"merge lists"
(6SBP INSOOe#$ 5SBP INSmore or no?$
  2SBP INSGlosses$ MBL cv DFL MON cr DBL
  (5SBP IMM pimore$? 3SBP INSview$ MBL cv DFL MON cr DBL
    6SBP MFC INC 3SBP SAX 2SBP
    (pm? MON 6EXC CPL TRA DFL TRA:);;)
  5SBP DFL 6SBP DFL 2SBP
  (pm? MEL:;) INS#e$
  2SBP)#

(Initialize the nnMFL# subroutine. Initialize the prompt
question. Read the gloss file into buffer 2.
  (Merge another file? Read it into buffer 3. Increment the
    nnMFL# subroutine.
    (Merge the list in buffer 3 into the merged master file
      in buffer 2.);)
  Clear buffers 5 and 6.
  (Find the end of the merged master file.) Append #e as end
  Display results in buffer 2.) marker.
```

## 4.2 Phonemic analysis

The buffer assignments for the phonemic analysis programs are the following:

- Buffer 1 - Programs
- Buffer 2 - Gloss file
- Buffer 3 - Combined word list and gloss file
- Buffer 4 - Words alone; translation of words to CV shapes
- Buffer 5 - Phone occurrences count; CV translation table
- Buffer 6 - Syllabification rules
- Buffer 7 - Results buffer
- Buffer 8 - Extracted list of "find examples"

## 4.2.1 Words alone

"words alone"

```
(4SBP DFL
  (pm? DFC MON MFL:;)
  4SBP)#
```

(Delete the mnemonic identifier.  
 (Delete the cognate set identifiers.)  
 Display the results.)

## 4.2.2 Phone occurrences

"phone occurrences"

```
(5SBP (pm? DFL:;)
  4SBP (SAX MON 5SBP
    (pe? 4MFC INC; 6MFC pm?:; CPC INS_001$)
    TRA MFC pm?:;)
  5SBP (pi$? DEC INS#$; MFL:)
  5SBP (SAX 5SBP
    (pa? MFL; pl? MFL:; CPL TRA DFL)
    MON pm?:;)
  5SBP)#
```

((Clear the results buffer.)

Set BP to data buffer.

(Set Aux to current character in data, BP to results buf.

(Is this the current character?

Yes, move to the count and increment;

No, move to next entry. More in table? Yes, repeat:

No, make a new entry. Copy character and  
 initialize count to 001.)

Return to data and advance to next character. More data?:;)

(Change CRLF in table to #--a printable character.)

(Sort the table into alphabetical order of the phones.

See PTP user's manual, section 6.1.2.)

Display the results in buffer 5.)

## 4.2.3 Phone co-occurrences

"phone co-occurrences"

```
(7SBP (pm? DFL:;))
4SBP INS$ MBC
(SAX MON 7SBP
(2pe? 5MFC INC; 7MFC pm?;; 2CPC INS_001$)
TRA 2MFC pm? MBC:;))
4SBP DFC 7SBP
(pm? (pie$? DFC INS#$; MFC) INS-$
(pie$? DFC INS#$;) MFL:;))
7SBP (SAX 7SBP
(pa? MFL; pl? MFL:; CPL TRA DFL)
MON pm?;;)
7SBP)#
```

The structured program description is the same as for the previous program. The main difference is that the entries in the results table are two characters instead of one.

## 4.2.4 Word shapes

"word shapes"

```
(7SBP (pm? DFL:;))
4SBP (SAX 5SBP
(pe? MFC TRA CPC DFC MON; MFL:) (pie$? MFC;) pm?;;)
4SBP (SAX 6SBP SSV MFC TRA pm?
(kpe? TRA kMFC SSV TRA kMFC INS.$;
TRA MFL pm? SSV MFC TRA:; TRA MFC):;))
4SBP (SAX MON 7SBP
(pf? MFL 2MFC INC; 2MFL pm?;; CPL INSOOL$)
TRA MFL pm?;;)
7SBP (pm? MFL DBC INS_$ MFL:;)
7SBP (SAX 7SBP
(pa? MFL; pl? MFL:; CPL TRA DFL)
MON pm?;;)
7SBP)#
```

```
( (Clear the results buffer.)
( Set Aux on data character and BP at beginning of translation
table. (Find the character in the table and translate it.)
(If the character is CRLF, let it be.) More data?;;)
Syllabification. (Set Aux on rules, BP on data. More data?
(Does left part of rule fit on current data?
Yes, insert the syllable break;
No, more rules? Get next rule;;
Advance to next character in data):;)
(Count word shapes. Table entries are two lines, shape and count.)
(Format the results. Reduce two lines to one line.)
(Sort the results buffer) Display results.)
```

## 4.2.5 Syllable shapes

"syllable shapes"

```
(7SBP (pm? DFL:;))
4SBP INS#$
      (ps$? DBC INS#.$$;;)
4SBP (SAX ps.$? MBC INS$ 7SBP
      (pf? MFL 2MFC INC; 2MFL pm?;; CPL INSO01$))
      TRA ps$ DBC MFC:;)
4SBP DFC
      (ps#.$$? 3DBC INS$;;)
7SBP (pm? MFL DBC INS_ $ MFL:;)
7SBP (SAX 7SBP
      (pa? MFL; pl? MFL:; CPL TRA DFL)
      MON pm?;;)
7SBP)#
```

( (Clear results buffer.)

Put an initial word boundary at beginning of data.

(Replace all CRLF's with #.#--word final, syllable break,  
word initial.)

Go to beginning of data.

(Set Aux at beginning of current syllable. Find next  
syllable break and insert CRLF before it, so we can use  
pf for searching in the results table. Got to results table.

(The table entries are two lines. First line is syllable  
shape, second is count. Tabulate the current syllable)

Go back to data and delete the CRLF. Next syllable:;)

Delete the initial word boundary at the beginning of the data.

(Replace the #.#'s with CRLF's.)

(Format Results. Reduce two line entry to one line.)

(Sort results by alphabetical order of syllable shapes.)

Display the result buffer.)

## 4.2.6 Find examples

"find examples"

```
(8SBP SSV 3SBP MFL SAX 4SBP
      (pm? (pi$? TRA MON MFL TRA;))
      (pi--$? TRA SVL TRA;))
      MFC:;)
8SBP (pm? DFC MON MFL:;)
8SBP)#
```

(Set Save to result buffer. Go to combined word list/gloss file,  
move past mnemonic identifier, and set Aux. Go to data buffer.

(More? (Is character CRLF? Go to combined file and advance line;))

(Found an example? Go to combined file and save example;)

Advance to next character in data buffer:;)

(Strip cognate set identifiers from examples in result buffer.)

Display results in buffer 8.)

### 4.3 Lexicostatistic analysis

The buffer assignments for the lexicostatistic analysis programs are the following:

Buffer 1 - Programs  
 Buffer 2 - Merged master file  
 Buffer 3 - Unused  
 Buffer 4 - Table of cognate counts  
 Buffer 5 - Unused  
 Buffer 6 - Cognate matrix  
 Buffer 7 - nnMFL# subroutine

#### 4.3.1 Count cognates

"count cognates"

(4SBP SSV 2SBP MFL

(pi##?; SAX (MFL MON pi##? TRA MFL; TRA SVL TRA SVL:):)

4SBP

(pm? INS000 \$ ps\$ DBC INS-\$ MFL:;)

(4SBP SSV TRA MON MFL pm?

(pi##?; SAX

(MFL pi##? TRA MFL;

(pc? TRS 2MFC INC; TRS) MFL TRS:):);)

4SBP)#

(First put the mnemonic identifiers for each pairwise comparison into the result buffer. Set Save to result buffer. Set BP to merged master file and advance past gloss file mnemonic.

(End of mnemonics?; Set Aux to current mnemonic, then compare it to all the remaining mnemonics.

(Advance to next mnemonic. End of mnemonics?

Yes, go back to current mnemonic and advance it;

No, save the two mnemonics into the results buffer:))

(Format the results table. Insert the initialized count 000 as first item in each entry. Put a hyphen between the two mnemonics or the lists being compared in that line.)

(The cognate counting follows the same basic strategy as the routine above for copying the mnemonics into the results buffer. Each word list item is taken in turn with the next gloss, #, being the end mark for an item. The lists are compared in a lower triangle fashion in the same order as the mnemonics were copied into the results buffer. If the cognate set identifiers for two words are the same, the count is incremented. In either case, every time a comparison is made, the Save pointer is advanced to the next entry in the results buffer.)

Display the table of cognate counts.)

## 4.3.2 Build cognate matrix

```

"build cognate matrix"
(7SBP INS00e#$ 6SBP INS0$ SSV
 2SBP MFL
  (pi#$?; SVL MFL:)
 4SBP 4MFC
  (pm? SAX 7SBP MFC INC TRA SAX 6SBP MON 7EXC SSV TRA SAX
   (MBL TRS 2MFL MBC TRS 3SVC MFL 4MFC 3pe?:;):;))
6SBP INS***$ SSV 2SBP MFL (pi#$?; 3SVC MFL:)
4SBP 4MFC
  (SAX TRS 2MFL MBC INS***$ SSV TRA SAX
   pm? (MBL 3SVC MFL 4MFC 3pe?:;):;))
6SBP
  (pm? (3MFC INS pi$? MFC;:):;))
7SBP DFL 6SBP)#

```

(Initialize the nnMFL# subroutine. Insert a CRLF to terminate the first row of the matrix.

(Copy all the mnemonics as separate lines--first item in each row)

(Fill in the figures for the lower triangle by columns left to right.

First increment the nnMFL# subroutine. This routine is used to specify the row in which the next column begins.

Execute the nnMFL# subroutine to set Save at the proper row for beginning the column. Aux is set to the mnemonic for the column in the cognate count table in 4.

(As long as the first mnemonic in the cognate table equals the mnemonic for the column, copy the number of cognates at the end of each row.))

(Copy the mnemonics from buffer 2 to fill in the first row.)

(Now fill in the diagonal and the upper triangle from left to right and top to bottom.

(As long as the first mnemonic in the cognate table equals the mnemonic for the current row, copy the figures.))

(Insert a space between all the columns.))

## 4.3.3 Permute matrix

```

"permute matrix"
(7SBP INS00e#$
 6SBP (SAX 7SBP MON MFC INC TRA pi$?; MFC:)
   (6SBP IMM pi*?; INS$ TRA INS0$ 6SBP SAX
    (TRA MFL TRA MFC 3pe?:; TRA SSV TRA MFC)
    (TRA MFL TRA 4MFC 3pe?:; (pi$?; TRS TRA TRS) TRA SVL DFL)
  6SBP (4MFC pi$? SSV DFC ps$ DBC TRS
    (4SVC 4DFC 7EXC TRS 7EXC TRS pm?:;);
    pi$? SSV DFC ps$ DBC
    (4SVC 4DFC 4MBC 7EXC TRS 4MBC 7EXC TRS pm?:;):;))
7SBP DFL 6SBP)#

```

(Initialize the nnMFC# subroutine.

(Increment the nnMFC# command for every character in a row of the matrix. That is, set the argument of MFC to the number of characters in a row of the matrix.)

(Set BP to upper left corner of matrix. Go to immediate mode.

Terminate?; Insert delta for destination at BP. Insert sigma for source at Aux. Move the source row to destination.

(Find the row corresponding to the first column marked as source or destination and set Save pointer at it.

3pe fails when the first sigma or delta is found)

(Find the row corresponding to the other column involved.

(Is it the source row?; Trade the Save pointer)

Save the source row to the destination and delete it.)

→ (Now move the source column.

Advance to next column. Is it the source column?

Yes, find the destination column. Set Save to destination and BP to source. (Move the source column row by row. Save the four characters, three digits and space, and then delete them. Move the Save and BP to the corresponding place in the next row by executing the nnMFC# subroutine);

No. Is it the destination column?

Yes, set Save. Find source column and set BP.

(Move source column row by row. Save the four characters and delete them. To move pointers to corresponding place in next row, first move back 4 characters to compensate for the deleted characters, then execute 7);

No:)

Go back to immediate mode to display results and do it again:)

Clear buffer 7)

#### 4.4 Lexical isogloss analysis

The buffer assignments for the lexical isogloss analysis programs are the following:

Buffer 1 - Programs  
 Buffer 2 - Merged master file  
 Buffer 3 - Unused  
 Buffer 4 - Cognate counts  
 Buffer 5 - Lexical isogloss file  
 Buffer 6 - Cognate matrix  
 Buffer 7 - nnMFC# subroutine

## 4.4.1 Lexical isoglosses

*add MON's**tape uses buffer 3***"lexical isoglosses"**

(5SBP SSV 2SBP

(pm?^(pi#\$\$? SVL;) MFL;; TRS DBL)

5SBP SSV 2SBP

(MFL pm?^(SVC MFL pi#\$\$? TRS MFL TRS;:);)

5SBP)#

(Set Save to buffer 5, set BP to merged master file.

(On the first pass through the data save all the gloss entries.

After all the glosses are saved, delete the last one which  
was only the end marker, #c.)

Reset the pointers.

(On the second pass through the data, save all the cognate  
set identifiers in order in front of the proper gloss.)

Display the results.)

## 4.4.2 Isogloss analysis

**"isogloss analysis"**

((5SBP SAX IMM pa?;SAX 7SBP INSOod## 3MBC SSV 5SBP

(pa?; TRS INC TRS MFC:)

5SBP (pm? MON SSV 7EXC SVC DFC MFL:;) 7SBP DFL:)

5SBP INSOα\$ MBC

(SAX 5SBP

(pa? MFL; pl? MFL;; CPL TRA DFL)

MON pm?;:)

5SBP DFC)#

*(missing on tape)*

(First put the lists into the desired order.

(Set Aux to beginning of isogloss file and go to immediate mode.

Is the user through moving lists? Yes, go to sort;

No, initialize an nnMFC# subroutine.

(Set the nn in subroutine equal to the number of lists in  
front of the one being moved.)(Go through the whole isogloss file saving the  
element from the proper list to the front of the line and  
deleting it from its original position.)

Zero the nnMFC# subroutine and go back to beginning:)

Sort the isogloss file. First put an alpha in front of the  
first line containing the one character identifiers for the  
lists. This ensures that the identifier line will be the  
first line in the sorted file.

(Sort the file.)

Delete the alpha and display the results.)



#### 4.5 Phonostatistic analysis

The buffer assignments for the phonostatistic analysis programs are the following:

- Buffer 1 - Programs
- Buffer 2 - Merged master file (formatted)
- Buffer 3 - Degrees of difference table
- Buffer 4 - Results for each pairwise phonostatistics comparison
- Buffer 5 - Alternated word list for two lists being compared
- Buffer 6 - nnMFL# subroutine

##### 4.5.1 Check format

"check format"

```
(6SBP INSOOe#$ 3MBC SSV 2SBP (MFL TRS INC TRS pi#$$?;:))
3SBP (pm? DFL:;)
5SBP SSV 2SBP MFL
  (INS**$ pi#$$? 2DBC;
    (2SBP ps*$ SAX ps*$ DBC MFL INS*$ pi#$$? DBC 2SBP ps*$ DBC MFL;
      (TRA 6EXC TRA 6EXC pm?
        (pe? TRA MFC TRA MFC
          ((p1? TRA;) TRA SVC pi#$$? TRA
            (pi#$$?; SSV 3SBP (pm? DFL:;) 5SBP (pm? DFL:;)
              6SBP DFL 2SBP ps*$ DBC ps*$ DBC
                TRS INS**ERROR**$ IMM)
              SVC; MFC TRA SVC MFC:;) MON:;)
          5SBP
            (SAX (pi#$$?; pi-$?; MFC pi-$? MBC; MBC pi//$?;
              3SBP (2pe?; 2MFC pm?:; 2CPC) TRA) 2DFC MON pm?:;))
            :):)
3SBP (pm? 2MFC INSc$;;)
6SBP DFL 2SBP)#
```

(Initialize an nnMFL# subroutine.

(Set nn to number of lists including the gloss list.)

(Clear buffer 3.)

Set Save to buffer 5 for the alternated word list. In the alternated list, the two word lists are alternated character by character as they are saved. That is, first a character from list one, then a character from list two, one from list one, one from list two, and so on. The result is a series of sound correspondences.

Go to beginning of merged master file. Advance past mnemonic for the gloss file. In the following routine, asterisks are used to mark the mnemonic identifiers of the last two lists which were compared.

```

(Insert **. Have we done all pairs? Yes, delete **. Exit;
(Set Aux to first * and advance the second *.
  Is it to end of mnemonics? Yes, delete second *.
  Find first * and delete it. Advance to next mnemonic;
  No, extract an alternated file of the current two lists.
  (Advance to next pair of words. More in file?
  (Are the forms cognate? Move past cognate set IDs.
    ( (We want to store only a triangular matrix since
      degrees of difference is symmetrical. Set Aux
      to the character that is lexically greater.)
      Save the first character. Is it CRLF?
      YES, (Is other character CRLF? Yes, go ahead;
      No. Formatting error. Clear buffers 3,
      5, and 6. Delete '*'s from buffer 2.
      Insert **ERROR** message. Call immediate
      mode to terminate.)
      Advance to next character. Save and advance
      second character.:););)
Alternated list is in buffer 5. Now look up all the sound
correspondences to make sure they are in the degrees of
difference table listing.
  ( (Take a correspondence from the alternated list.
    If it is end of word, contains a -, or // there is
    no correspondence. Is it no correspondence?;
    (Look up the correspondence in the list in buffer 3.
    If it is not there, add it.);)
    Delete the correspondence from alternated file. Advance.
    More correspondences to look up?;);)
:);)
(Insert a CRLF after every correspondence in the table in 3)
Clear buffer 6 and display the merged master file.)

```

#### 4.5.2 Phonostatistics

"phonostatistics"

```
(6SBP INS00e#$ 3MBC SSV 2SBP (MFL TRS INC TRS pi#$?;:)
```

```
2SBP MFL
```

```
(INS**$ pi#$? 2DBC;
```

```
(2SBP ps*$ SAX ps*$ DBC MFL INS*$ pi#$? DBC 2SBP ps*$ DBC MFL;
```

```
SSV 4SBP TRS TRA SVL TRS DBC INS-$ TRS TRA SVL SSV 5SBP TRS
```

```
(TRA 6EXC TRA 6EXC pm?
```

```
(pe? TRA MFC TRA MFC
```

```
((pl? TRA;) TRA SVC pi#$? TRA SVC; MFC TRA SVC MFC:);)
```

```
MON;);
```

```
4SBP MFL INS w_000e c_000e d_000e 0_000e 1_000e 2_000e 3_000e
```

```
4_000e 5_000e $ 5SBP
```

```
(SAX (pi#$? 4SBP MFL 4MFC INC; pi-$?; MFC pi-$? MBC; MBC pi//$?;
```

```
3SBP (2pe? 2MFC SSV 4SBP 2MFL 4MFC INC 6MFC kINC
```

```
MFL kMFL 4MFC INC; MFL:))
```

```
TRA 2DFC MON pm?;);
```

```
4SBP IMM 4SBP 10DFL:);) 6SBP DFL 2SBP)#
```

The initialization routine and the basic structure of the program are the same as the "check format" program. The fifth line is completely new. It puts the title line into the results buffer, eg. MNE<sub>1</sub>-MNE<sub>2</sub>. The alternated word list for each pair of lists is extracted into buffer 5 as before. The code for the look up and result counting is new. A description is as follows:

```
Initialize the results buffer.
((Get next correspondence. Is it end of a word?
  Count a word in the results buffer;
  Is first character -?; Is second character -?; Is it //?;
  Look up correspondence in the degrees of difference table.
  (Found? Increment correspondence count. Increment sum of
   d. of d. count. Go to line for k d. of d. and increment;
   No, advance to next entry in d. of d. table:))
Delete current correspondence. More?;))
Display results in immediate mode. Zero results buffer on
return to program control. Go back and do next pair of lists.
```

#### 4.6 Comparative method

Buffer assignments for the comparative method programs are the following:

```
Buffer 1 - Programs
Buffer 2 - Formatted merged master file; reduced to a list
           of glosses only in "count correspondence sets"
Buffer 3 - Degrees of difference table remains
Buffer 4 - Alternated list for all of the individual lists
Buffer 5 - Count of correspondence sets
Buffer 6 - nnMFC# subroutine for "find examples" and
           "count correspondence sets"
Buffer 7 - nnMFC# subroutine for "isogloss analysis"
Buffer 8 - Extracted list of examples for "find examples"
```

##### 4.6.1 Prepare a set

```
"prepare a set"
(TRA pi#$$? (MON MFL pi#$$?; DFC:);
SAX (MFL pi#$$?; (pe? DFC; (DFC pi#$$?;:))):)
TRA SSV (MFL pi#$$?; SVL)
TRS MBL (pi#$$?; DFC INS-$.)
MBL SAX 2MFL (pi#$$?; pi#$$? CPL DFC; MFL:)
TRA DFL TRA)#
```

(Is there no cognate set identifier?  
 Yes, (Delete all cognate set identifiers for the set.);  
 No, there is a cognate set identifier. Set Aux to the ID.  
 (Advance to next line. End of set?  
 (Is this word in the right cognate set? Delete identifier;  
 No, (Delete word, leaving only CRLF)):  
 Set Save to precede the gloss.  
 (Find first word in the set which is the proper cognate set.  
 Save the word.)  
 (Convert the saved word to a string of hyphens. This gives  
 a string of hyphens the same length as all words in the set)  
 Set Aux to string of hyphens. Move to first word entry.  
 (End of set?; Is the entry CRLF only? Copy the string of  
 hyphens and delete the CRLF; Advance to next entry:)  
 Delete string of hyphens. Set BP to next gloss.)

#### 4.6.2 Delete a set

"delete a set"  
 (TRA (MON DFL pi#\$\$?;:))#  
 (Delete all the lines between Aux and the next #.)

#### 4.6.3 Copy a set

"copy a set"  
 (TRA SSV (MFL pi#\$\$?;:))  
 TRS SAX (MON SVL MFL pi#\$\$?;:) TRA)#  
 ( (Find the end of the set.) Set Save to end of set.  
 (Save the set.) Set BP back to beginning of original set.)

#### 4.6.4 Count correspondence sets

"count correspondence sets"  
 (5SBP SSV 2SBP  
 (MFL (pi#\$\$?; MFC pi#\$\$?; MBC)? TRS INS\_htd\$ TRS; SVC:)  
 SSV 4SBP TRS  
 (MON MFL pm?  
 (SAX (SVC DFC MFL (pi#\$\$?; MFC pi#\$\$?; MBC)?;:)  
 TRA pi\$? SVC ( DFC pi\$?;:);:);:)  
 6SBP INS00d#\$ 3MBC SSV 2SBP  
 (DFL (pi#\$\$?; MFC pi#\$\$?; MBC)?; TRS INC TRS:)  
 6SBP SSV 4SBP  
 (SAX MON 5SBP  
 (2kpe? 2kMFC 3MFC INC; MFL pm?;: 2kCPC INS\_00l\$)  
 TRA 2kMFC (pi\$? MFC;) pm?;:)  
 5SBP)#

The program makes use of a compound predicate--  
 (pi#\$\$?; MFC pi#\$\$?; MBC)? --to find the end of a set. After  
 formatting, the gloss line may begin with # or it may begin with  
 a cognate set identifier in which case the # is the second  
 character. The compound predicate tests for a # in the first or  
 second position. On falling through the right parenthesis, the  
 predicate register contains the truth value of the last predicate  
 which was executed. Thus the compound predicate will return true  
 if the # was found, false otherwise.

((First put the line of one character identifiers into result buffer)  
 Extract the alternated list.

(Move to first word in a set. More data in file?

( (Save and delete first character left in each word.

Makes a correspondence set in the alternated list.)

Are we to ends of the words? Save one CRLF then delete

them all.; Go back and save another correspondence:);).

Initialize a nnMFC# subroutine. The subroutine is used in

"find examples". The nn number only is used in this program

with the k command. (Set nn to number of lists.)

Count the correspondences. Set Save on number of lists, BP  
 on alternated list.

(Set Aux on alternated list. Go to results buffer.

Look up the current correspondence.

(Found? Increment count; advance in table. More entries;;

Copy the correspondence into the table and init count)

Advance to next correspondence

(If next character is CRLF move past it) More data?;;)

Display results.)#

#### 4.6.5 Isogloss analysis

This program is identical to the "isogloss analysis" program  
 in 4.4.2.

#### 4.6.6 Find examples

"find examples"

(8SBP SSV 2SBP SAX 4SBP

(pm? (pi--\$\$? TRA SVL TRA;)

6EXC (pic\$\$? MFC TRA MON MFL TRA;);)

8SBP)#

(Set Save to result buffer. Set Aux to list of glosses.

Set BP to alternated list.

(More in the list? (Is this an example of the correspondence?

Go to gloss list and save example;)

Advance to next correspondence.

(Is it CRLF? Advance past it, and advance a line in gloss file;)

::)

Display results in buffer 8)

## 4.7 Refined phonostatistic analysis

The buffer assignments for the refined phonostatistics program are the following:

- Buffer 1 - Programs
- Buffer 2 - Formatted merged master list
- Buffer 3 - Degrees of difference table
- Buffer 4 - Results for each pairwise phonostatistic comparison
- Buffer 5 - File of regular correspondences
- Buffer 6 - Alternated word list for the two lists being compared
- Buffer 7 - List of regular correspondences for the two lists
- Buffer 8 - nnMFL# subroutine, where nn is number of lists plus one
- Buffer 9 - nnMFC# subroutine, where nn is the number of characters in a correspondence set entry in 5
- Buffer 10 - A 4MFC# or 8MFC# subroutine. This subroutine is used to position the BP to the proper column in the results table--4MFC for regular, 8MFC irregular
- Buffer 11 - Flag for signalling whether or not a word contains any irregular correspondences. The flag is initialized to zero before each new word. Any time an irregular correspondence is encountered the flag is incremented.

"refined phonostatistics"

```
(8SBP INSO0e#$ 3MBC SSV 2SBP (MFL TRS INC TRS pi#$$?;:))
9SBP TRS MBC 2SVC TRS MBC 4INC MFC INSD#$ 10SBP INSD#$
11SBP INSO$ 5SBP INS**$ 2SBP MFL
  (INS**$ pi#$$? 2DBC 5SBP ps*$ DBC ps*$ DBC;
  (2SBP ps*$ SAX ps*$ DBC MFL INS*$ pi#$$? DBC 2SBP ps*$ DBC MFL;
  SSV 4SBP TRS TRA SVL TRS DBC INS-$ TRS TRA SVL SSV 6SBP TRS
  (TRA 8EXC TRA 8EXC pm?
  (pe? TRA MFC TRA MFC
  (TRA SVC pie$? TRA SVC; MFC TRA SVC MFC:;)) MON:;))
7SBP SSV
  (5SBP ps*$ SAX ps*$ DBC MFC INS*$ pi_$?
  DBC MBL ps*$ DBC MFC INS**$; TRA MFC TRA
  (TRA 9EXC TRA 9EXC MON pm? TRA SVC TRA SVC:;))
4SBP MFL INSw 000 000e2 000 000e3 000 000e4 000 000e5 000 000e6
1_000_000e2_000_000e3_000_000e4_000_000e5_000_000e6$ 6SBP
  (SAX (pie$? 4SBP MFL 4MFC SSV 11SBP
  (pio$? TRS; DFC INSO$ TRS 4MFC) INC;
  pi-$?; MFC pi-$? MBC; MBC pi//$?; 10SBP DFC 7SBP
  (2pe? 10SBP INS4$; 2MFC pm?; 10SBP INS8$ 11SBP INC)
  (TRA SAX MFC pe?; pl? MFC CPC TRA DFC SAX;))
3SBP (2pe? 2MFC SSV 4SBP 2MFL 10EXC INC MFL
  10EXC kINC MFL kMFL 10EXC INC; MFL:))
TRA 2DFC MON pm?;))
4SBP IMM 4SBP 10DFL 7SBP DFL:;))
8SBP DFL 9SBP DFL 10SBP DFL 11SBP DFL 2SBP)#
```

(Initialize buffers 8 to 11. Asterisks are used to mark the lists currently being compared, as already detailed in "check format". Insert \*\* in list of regular correspondences. Go to master file.

(Insert \*\*. Have we done all pairs of lists? Yes, exit;

(Set Aux at first \*. Advance the second \*. Is it to end?

Yes, delete it. Find first \*, delete it, advance to next line;

No. Save mnemonics of lists being compared into results buffer.

Extract an alternated list of the current two lists into 6.

(Advance to next pair of words. More in file?

(Are the forms cognate? Move past cognate set identifiers.

(Save character from first list. Is it CRLF?

Yes, save CRLF from second list. Do not advance pointers;

No, save character from second list. Advance ptrs:););)

Extract the list of regular correspondences for the two lists into 7.

(Set Aux to first \*. Advance second \*. End of mnemonics?

Yes, delete second \*, delete first \* and insert \*\* in next position;; No. Advance BP one character past first \* to counteract having to move forward an extra character past the second \*.

(Advance to next correspondence set. More in file?

Save the correspondence for the two lists:);)

Initialize the results buffer.

Go to the alternated list in 6 and tabulate the results.

((End of a word? (Check the flag in 11 for any irregulars. Advance to proper column in results table) Count the word;

Does the correspondence contain - or //? No comparison;

Initialize the MFC subroutine in buffer 10. See if the current correspondence is in the list of regular correspondences in buffer 7.

(Is it this correspondence? Yes, set buf 10 to 4MFC;

No. Advance to next regular correspondence. More?;;

It is an irregular correspondence. Set buf 10 to 8MFC.

Increment the flag in buffer 11.)

(The degrees of difference is a triangle matrix--that is, the second character in the entry is always equal to or lexically greater than the first character. Thus the current correspondence must be checked. If the first character is lexically greater than the second, the characters must be swapped.)

Find the current correspondence in the degrees of difference table.

(Found? Increment correspondence count. Increment sum of d. of d. count. Go to line for k d. of d. and increment. In all cases, the 4MFC or 8MFC in buf 10 is executed to increment the proper column in the table; Not found, move to next entry in d. of d. table:))

Delete the current correspondence and advance to next. More?;;)

Display results in immediate mode. On return to program mode, delete results in buf 4 and list of regular corr's in 7:);)

Clear buffers 8 through 11. Display merged master list.)

## REFERENCES

- Carroll, John B. Isidore Dyen. 1962. High speed computation of lexicostatistical indices. *Language* 38:274-78.
- Frantz, Donald G. 1970. A PL/I program to assist the comparative linguist. *Communications of the ACM* 13:353-6.
- Grimes, Joseph E. 1964. Measures of linguistic divergence. *Proceedings of the ninth international congress of linguists*, edited by H.G. Lunt. The Hague: Mouton. pp. 44-50.
- \_\_\_\_\_ and Frederick Agard. 1959. Linguistic divergence in Romance. *Language* 35:598-604.
- Keesing, Roger and J. Fifi'i. 1969. Kwaio word tabooing in its cultural context. *Journal of the Polynesian Society* 78:154-77.
- Ladefoged, Peter. 1968. The measurement of cross-language communication. Uganda language survey working paper. Educational resources information center document ED 024 920 (AL 001 284). Washington: United States Office of Health, Education and Welfare.
- McKaughan, Howard. 1964. A study of divergence in four New Guinea languages. *American Anthropologist* 66:98-120.
- Sanders, Arden G. 1977. Guidelines for conducting a lexicostatistic survey in Papua New Guinea. In *Language Variation and survey techniques, Workpapers in Papua New Guinea languages*, Vol. 21. Ukarumpa, E.H.P.: Summer Institute of Linguistics. pp. 21-41.
- Simons, Gary. 1977. A user's manual for PTP--the Programmable Text Processor. Working papers for the language variation and limits to communication project, Number 1. Ithaca, NY: Cornell University.
- \_\_\_\_\_. 1977a. Phonostatistic methods. *Workpapers in Papua New Guinea languages* 21:155-185.
- \_\_\_\_\_. 1977b. Recognizing patterns of divergence and convergence in a matrix of lexicostatistic relations. *Workpapers in Papua New Guinea languages* 21:107-134.
- \_\_\_\_\_ and Linda Simons. 1977. A vocabulary of Biliu, an Austronesian language of New Guinea, with notes on its development from Proto Oceanic. Working Papers for the language variation and limits to communication project, Number 2.