RATS: A Middle-Level Text Utility System

John B. Smith

The Random-Accessible Text Systems (RATS) is an intermediate level utility, written in PL/I, for the researcher who has some knowledge of programming.[1] It provides a highly flexible representation of a text, allowing the researcher to write specific programs to perform the exact analysis he desires. There are, of course, a number of natural-language utilities available, but often they do not fit exactly the user's research design. He must then either modify his analysis or set out to develop his own system. Such "rediscovery of the wheel" is sometimes educational but unnecessary. The RATS system allows the user to bypass most of the "housekeeping" steps of file creation and begin with the actual analytic steps. Thus, he can concentrate on the more important phases of his project, thereby maintaining the integrity and sensitivity of his research design. This flexibility is achieved by breaking the text into three data sets linked by pointers, such that the user has access to the text in both alphabetical and text or linear order and may move easily from one to the other. Once the text is in the RATS structure, most of the programs that the user tailors to his needs are quite simple to write, as will be seen in the examples discussed below.

Input

RATS uses as its initial text scanner the INDEX program of S.Y. Sedelow's VIA-II. This scan program was chosen because of its availability, its relative ease of use, and the practicality of the information generated. I shan't attempt a complete description of it (see S.Y. Sedelow's *Automated Analysis of Language Style and Structure,* 1970, for a user's manual for this program); however, I shall describe briefly input and output format. Input is punched text in columns 1-72, in blank delimited form, that is, all words and punctuation marks are separated by blanks. Output is a token data set, one word per logical record, with two sets of index values: a linear number indicating text position (the first word is numbered 1, the second 2, etc.) and a set of hierarchical counters that indicate, for example, volume, chapter, paragraph, sentence, and word in sentence for prose texts and similar counters appropriate for other genres. RATS uses this data set, sorted in alphabetical order.

[1] These programs were developed in part through the assistance of an N.S.F. Institutional Grant. Coding was done by Nancy Abbatiello.

John B. Smith is an assistant professor of English and a research consultant with the Computation Center at the Pennsylvania State University.

Example: Punched data.

NOW IS THE TIME FOR ALL GOOD MEN TO COME TO THE AID OF THE PARTY .

Example: Ouput records from VIA-II INDEX program.

lin	vol	ch	para	sent	wis	page	idiom	prefix	length	word
1	1	1	1	1	1	1			3	NOW
2	1	1	1	1	2	1			2	IS
3	1	1	1	1	3	1			3	THE
4	1	1	1	1	4	1			4	TIME
5	1	1	1	1	5	1			3	FOR
6	1	1	1	1	6	1			3	ALL
7	1	1	1	1	7	1			4	GOOD
8	1	1	1	1	8	1			3	MEN
9	1	1	1	1	9	1			2	TO
10	1	1	1	1	10	1			4	COME
11	1	1	1	1	11	1			2	TO
12	1	1	1	1	12	1			3	THE
13	1	1	1	1	13	1			3	AID
14	1	1	1	1	14	1			2	OF
15	1	1	1	1	15	1			3	THE
16	1	1	1	1	16	1			5	PARTY
17	1	1	1	1	17	1			1	.

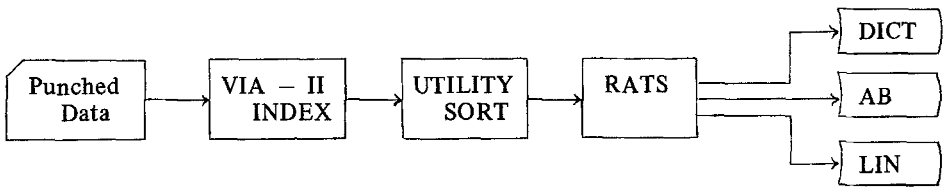
Data Structures

The token input set is built into three interconnected data sets that are stored on tape, disk, or some other auxiliary storage medium. The user then reads in the portions of these data sets appropriate for his particular analysis. The three data sets are:

1) DICTIONARY, a type file in which each word and punctuation mark is listed along with its frequency of occurrence in the text and a pointer to the second file where the index information for each occurrence is stored. These records are fixed length, with eighteen characters allotted for each text word. If this space is insufficient, it would be a simple programming problem to modify this; however, eighteen characters has been sufficient space for all texts that I have encountered.

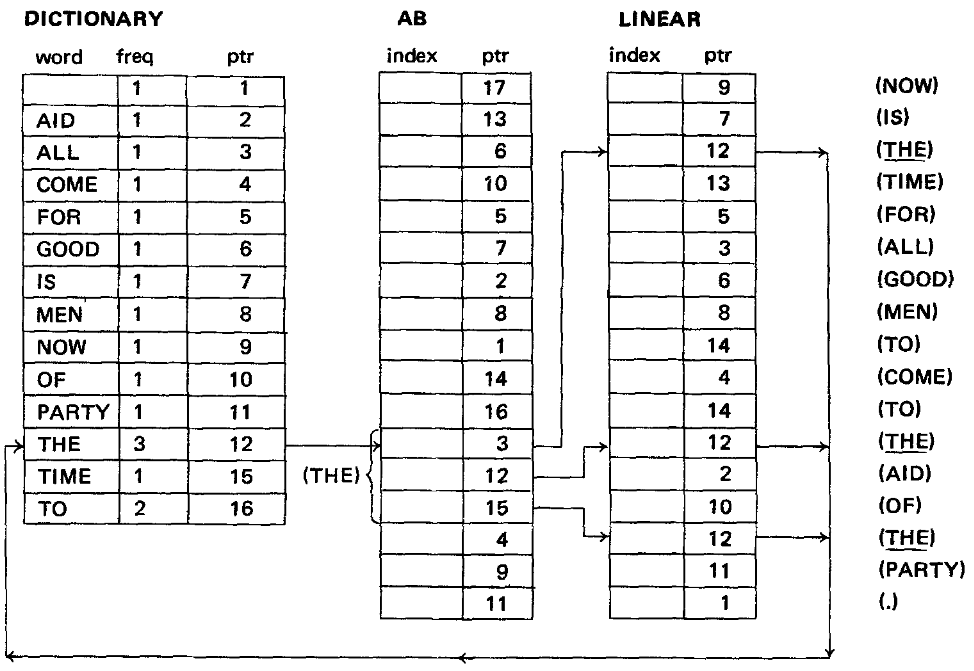
2) AB, a token data set of index information for each occurrence of each word in the text arranged in alphabetical order by word.

3) LIN, a token data set of index information for each occurrence of each word arranged in linear or text order. Instead of storing the word itself in each record, a pointer back to the dictionary indicates the word that appears at each linear position.



RATS Organization

Example: RATS file structure
Trace of the word, *the*



DICTIONARY

word	freq	ptr
	1	1
AID	1	2
ALL	1	3
COME	1	4
FOR	1	5
GOOD	1	6
IS	1	7
MEN	1	8
NOW	1	9
OF	1	10
PARTY	1	11
THE	3	12
TIME	1	15
TO	2	16

AB

index	ptr
	17
	13
	6
	10
	5
	7
	2
	8
	1
	14
	16
	3
	12
	15
	4
	9
	11

LINEAR

index	ptr	
	9	(NOW)
	7	(IS)
	12	(THE)
	13	(TIME)
	5	(FOR)
	3	(ALL)
	6	(GOOD)
	8	(MEN)
	14	(TO)
	4	(COME)
	14	(TO)
	12	(THE)
	2	(AID)
	10	(OF)
	12	(THE)
	11	(PARTY)
	1	(.)

Files two and three are linked by a pointer from file two to file three, thus completing the chaining among all three data sets.[2]

Programs

BUILD. This program takes the sorted token file from VIA-II INDEX and builds the data structure just discussed as three independent files: a dictionary, frequency, and pointer file; the alphabetically ordered file of index information and pointer; and the linearly ordered file of index information and word pointer back to the dictionary. This program is operational and available.

COMPRESS. This program takes the file structure created by BUILD and compresses it to either a linear subset of the text or a subset represented by a partitioning of the dictionary. Thus, a file structure for a chapter, for example, may be created so that the whole structure can fit into core. The second mode of compression allows the user to specify, say, the words he wishes to consider images or a theme and work with them only. By performing the analysis in logical modules the user can reduce costs in data handling and greatly simplify many of his programming problems that concern file manipulation. This program is operational and available.

EXPAND. This program does exactly the opposite of what COMPRESS does. Often it is desirable to process texts in small chunks and later amalgamate the results and/or data sets. EXPAND takes the file structures created by BUILD for several texts or subsets

[2] Of course, the index information attached to *both* files two and three is redundant, but since space on tape is relatively cheap it is included so that it will be available when a particular analysis requires the DICTIONARY and only one of the second or third files. It is unlikely that this index information would be stored within memory for both data sets at the same time.

of a text and combines them into a single RATS file structure. This program is expected to be operational by summer 1972.

A number of other functions and programs are planned, but discussion of them will be delayed until after the section concerning examples of application.

Examples of Application

Once a text is represented in the RATS file structures, many of the tasks that are commonly performed in textual analysis are easily programmed, but even more important, the representation affords such flexibility that the user may tailor his procedures to compute exactly the measures that fit his experimental model. To illustrate this, three applications are discussed briefly: a concordance procedure, a procedure to calculate the distribution of a "theme," and finally a procedure to construct the matrix of transitions among words over the text.

Concordance Construction. There have been, of course, many procedures to produce concordances; however, most of these do little more than produce the concordance and require that the text be punched with input conventions that make it unsuitable for other utilities. Since the user may wish to print the context for only a subset of words, like the imagery in a text or a thematic cluster, it is desirable for him to be able to easily tailor his concordance for his particular needs. If we look forward to the day of large-scale interactive systems, the user may eventually be able to bypass the voluminous output phase often associated with concordance construction and text analysis by having a program available that will produce the context information for only the words he is interested in when he needs it.

For this application most of the index information of the second and third files can be ignored. The user will read into PL/I structures the DICTIONARY file, the pointer in file two, and the pointer in file three. Depending on how he wishes to define the context of words, the user may ignore most of the index information in files two and three. If he wants, say, the five words on each side of the key word, then all of it can be ignored; if he wants the complete sentence, then the sentence-number within the index field of file three should be read in.

After the appropriate information from the files is read in, the main logic consists of only three nested DO-loops. The outermost goes from the first to last DICTIONARY entries. The second begins with the value of the pointer stored with the DICTIONARY and goes to (pointer_value + freq. -1) where freq. is the frequency for the particular DICTIONARY word-type. The third loop begins with the value stored in the pointer in file two and moves out on each side of that position in file three the desired number of locations. If the context is ±5 words, then the third loop goes from (pointer -5) to (pointer +5); if the context is the complete sentence, the DO-loop is executed as long as the sentence number is the same as that of the word for which context is being sought.

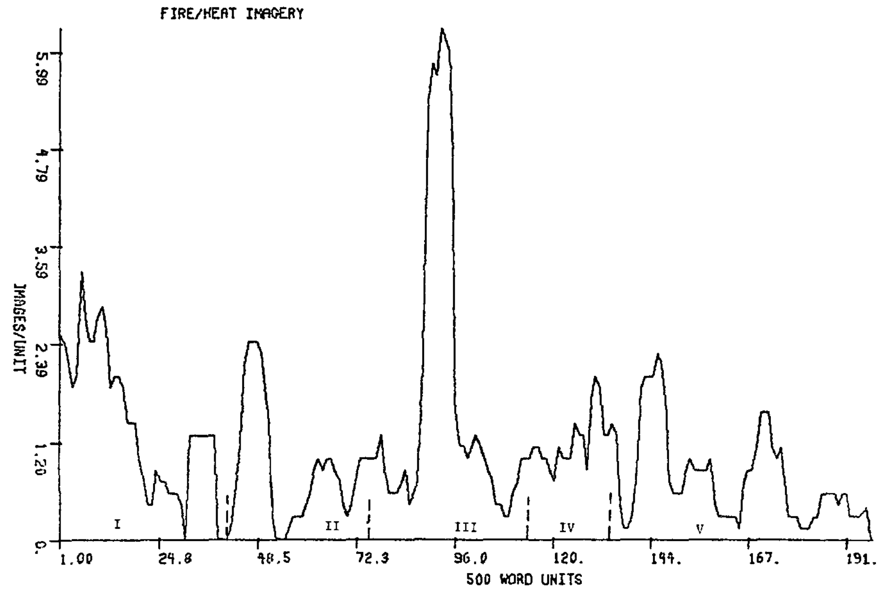
Basic PL/I statements for simplest concordance example:

```
DO I = 1 to DICTOP;
     DO J = DICT_PTR(I) to DICT_PTR(I) + DICT_FRQ(I)-1;
         DO K = AB_PTR(J)-5 TO AB_PTR(J) + 5;
            PUT EDIT (WORD (LIN_PTR(K))) (A);
          .
          .
          .
         END;
     END;
END;
```

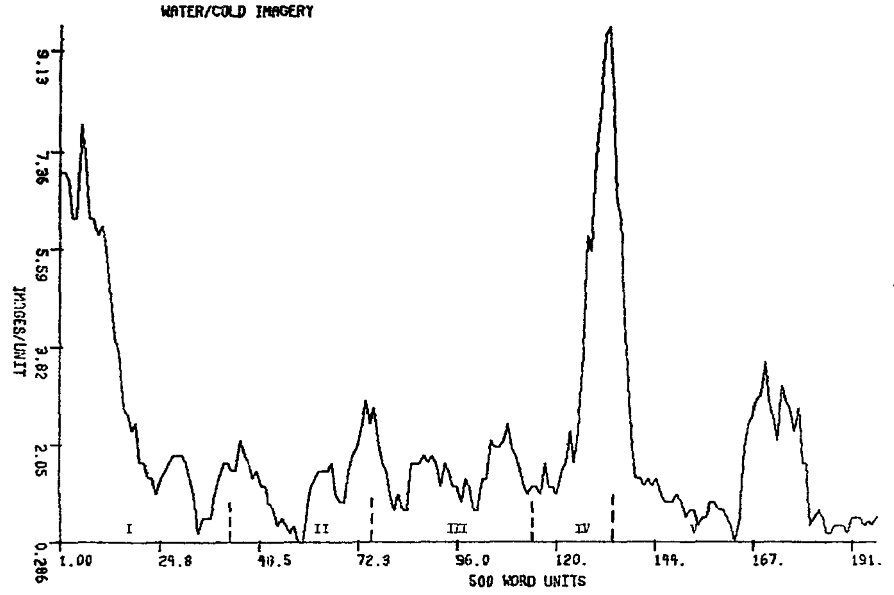
The user would, of course, have to test for top and bottom limits for K, and supply appropriate print statements, page control, etc.

Thematic Distribution. It is often quite helpful to represent the distribution of a theme by a line graph. For example, the smoothed distributions of fire and water imagery from Joyce's *Portrait of the Artist* given below show their "parallel" use in Chapter I and their respective dominance in Chapters III and IV.

There has been considerable discussion concerning the defining of theme; however, most methods boil down to defining it by semantic grouping, perhaps with the help of a computer-accessible thesaurus, context-clustering tendency, or by inspection. One



Fire/Heat Imagery



Water/Cold Imagery

interesting aspect of the RATS system is the clear distinction it makes between operative definition of a concept like theme and its interpretation, which is often mistaken as the definition itself. A thematic group of words is essentially a partitioning of the vocabulary; operatively, then, a theme is defined in these examples as a set of words that are denotatively similar; the fire/heat theme thus consists of images such as fires, flame, flames, burn, hot, etc. The distributions shown are the collective distribution of such sets of images. This partition can probably be most easily specified by a list of the index values of the words in the DICT file or by a logical vector, one entry per DICT item, with "1s" indicating inclusion and "0s" representing exclusion. With this approach it makes little difference where the partitioning came from or how it is interpreted: operatively, all such groupings function in the same way.

To compute a distribution the user establishes a unit interval size (perhaps 500 words), computes the number of such intervals within the text, and establishes a vector of counters—one for each interval. If, for example, his interval is defined in terms of words as opposed to, say, sentences, he needs only the DICT pointers and frequencies from file 1 and the linear numbers from file 2. If he is using a logical vector for the partitioning, his main logic reduces to two nested DO-loops similar to the outer loops for the concordance example. To compute the index value for the appropriate counter for each word-token, the linear number is divided by the unit size and 1 is added to the quotient ([lin#/unit size] + 1).
Example: for a unit of 500 words, word 1573 would fall into the fourth unit interval of the text (4 = (1573/500) + 1).

Basic logic in PL/I:

```
DO I = 1 to DICTOP;
      IF LOG_VECT(I) = '1' THEN
      DO J = DICT_PTR(I) TO DICT_PTR(I) +
            DICT_FRQ(I) -1;
            COUNTER ((LIN(J)/UNIT) + 1) =
                  COUNTER ((LIN(J)/UNIT) + 1) + 1;
      END;
END;
```

The resulting distribution, stored in COUNTER, can then be smoothed, if desired, and either printed out or passed directly to the plotter routines available at most computer centers.

Transition Matrix. The problem with computing a matrix of transitions or transition probabilities is the impossibility of storing in core a matrix of counters large enough for any sizeable text. RATS allows the user to bypass this problem entirely. The matrix is computed row-wise, and only as many counters as there are DICT items are required. The pointers in file three must be read in, and then files one and two are read in or passed sequentially. A bank of counters, one for each DICT item, is established. The main logic then consists of two nested DO-loops similar to those in the above examples. At the innermost level, the counter indexed by the succeeding word in file three is "bumped."

Basic logic in PL/I:

```
DO I = 1 to DICTOP;
   DO J = DICT_PTR(I) TO DICT_PTR(I) +
     DICT_FRQ(I) - 1;
       COUNTER (LIN_PTR(AB_PTR(J)+1))
       = COUNTER (LIN_PTR(AB_PTR(J)+1)) + 1;
   END;
    (OUTPUT COUNTERS OR PROBABILITIES).
END;
```

This output may either be printed or passed to some sort of cluster analysis or other analytic procedure. As before, the user must establish output procedures and appropriate tests for upper and lower limits and reinitialize counters.

Comments on Use

It is important to distinguish between data set format and the internal array format for core storage used by an actual program. Files two and three were created with redundant index information because in many applications only one or the other will be required. It is, thus, more convenient and cheaper to store the extra information than to have to read all three data sets to extract what is needed. Of course, the user will read in only the information in the file necessary for his analysis.

At this point it may be useful to comment on core requirements. Since the file structures described are obviously too large to be held in core in their entirety for any sizeable text, the user must be careful to allocate space for *only* the information he needs for a particular task. For example, if he is constructing a concordance, he will need the complete dictionary file at 24 bytes per entry, the pointers from the AB file at either 2 or 4 bytes per entry, and the pointer in the LIN file back to the DICTIONARY at 2 bytes per entry. However, the AB pointers will be used only in sequential order; so they may be read in, one at a time, as needed. Thus, for a two-hundred-page novel—probably in the neighborhood of 75,000 word tokens and some 4,000 to 5,000 word types—the storage requirements would be:

DICT	=	5k x 24 = 120k
LIN	=	75k x 2 = 150k
Total		= 270k

If this requirement exceeds available core, there are two obvious solutions to the problem. One, the DICTIONARY file can be put onto a random-access device and brought in as needed, or, better, the *word* in the DICT can be left out entirely, thus reducing the required space for the DICT from 120k to 30k (5k x 6 = 30k). Concordance entries, instead of being sent to the printer, would be composed not of words but of the DICT pointer values and sent to a direct-access device. Then, the DICT words in characters could be read in alone and the actual concordance lines constructed one at a time from the temporary file of word pointers just created.

This is an extreme example, however, for many applications do not need the word at all. The thematic example given above is such a case. Here, only the frequency and pointer values from the DICT file are needed. In fact, since this data is used sequentially, both the DICT and the AB file may be read in one value at a time, thus reducing the storage requirement to virtually nothing (10 bytes).

Because of the flexibility of RATS, these and other processing techniques soon became apparent to the user with several weeks' experience using the system.

Additional Programs

A number of additional functions and procedures are planned for the system, most of which should be operative by fall 1972. These include the following:

RANDOM. This program will load the data sets into random-access files for file structures too large for core.

RETURN/SEARCH. These functions will allow the user to specify the information he desires (RETURN) and then obtain that information by a SEARCH request defined on a particular word or group of words.

Conclusion

While this system does not supply all of the features we might wish, it does facilitate greatly many of the tasks that humanists and language analysts often perform. Its general flexibility should enable the user to design and implement procedures that fit his analytic model more precisely and sensitively than is generally possible. The results, hopefully, will be a greater number of computer-assisted works that are of definite substantive importance within the user's discipline.