DOCUMENT RESUME

ED 048 915 LI 002 724

TITLE	On-Line Retrieval System Design; Part V of Scientific Report No. ISR-18, Information Storage and Retrieval...
INSTITUTION	Cornell Univ., Ithaca, N.Y. Dept. of Computer Science.
SPONS AGENCY	National Library of Medicine (DHEW), Bethesda, Md.; National Science Foundation, Washington, D.C.
REPORT NO	ISR-18 [Part V]
PUB DATE	Oct 70
NOTE	100p.; Part of LI 002 719
EDRS PRICE	EDRS Price MF-$0.65 HC-$3.29
DESCRIPTORS	Automation, Computer Programs, *Design, *Information Retrieval, *Information Systems, Languages, Man Machine Systems, Programing, *Search Strategies, Shared Services, Systems Analysis, Use Studies
IDENTIFIERS	*Saltons Magical Automatic Retriever of Texts, SMART On Line Retrieval Systems

ABSTRACT

On-line retrieval system design is discussed in the two papers which make up Part Five of this report on Salton's Magical Automatic Retriever of Texts (SMART) project report. The first paper: "A Prototype On-Line Document Retrieval System" by D. Williamson and R. Williamson outlines a design for a SMART on-line document retrieval system using console initiated search and retrieval procedures. The conversational system is described as well as the program organization. The second paper: "Template Analysis in a Conversational System" by S. F. Weiss discusses natural language conversational systems. The use of natural language makes possible the implementation of a natural dialogue system, and renders the system available to a wide range of users. A set of goals for such a system is presented. An experimental conversational system is implemented using a template analysis process. A detailed discussion of both user and system performance is presented. (For the entire SMART project report see LI 002 719 and for parts 1-4 see LI 002 720 through LI 002 723.) (NH)

Department of Computer Science

Cornell University

Ithaca, New York 14850

On-Line Retrieval System Design Part V of

Scientific Report No. ISR-18

INFORMATION STORAGE AND RETRIEVAL

to

The National Science Foundation

and to

The National Library of Medicine

Reports on Analysis, Dictionary Construction, User Feedback, Clustering, and On-Line Retrieval

Ithaca, New York

October 1970

Gerard Salton

Project Director

SMART Project Staff

Robert Crawford
Barbara Galaska
Eileen Gudat
Marcia Kerchner
Ellen Lundell
Robert Peck
Jacob Razon
Gerard Salton
Donna Williamson
Robert Williamson
Steven Worona
Joel Zumoff

ERIC User Please Note:

This Table of Contents outlines all 5 parts of Information Storage and Retrieval (ISR-18), which is available in its entirety as LI 002 719. Only the papers from Part Five are reproduced here as LI 002 724. See LI 002 720 thru LI 002 723 for parts 1 - 4.

TABLE OF CONTENTS

PART ONE

AUTOMATIC CONTENT ANALYSIS

Available as LI 002 720

TABLE OF CONTENTS (continued)

TABLE OF CONTENTS (continued)

Page

PART TWO

AUTOMATIC DICTIONARY CONSTRUCTION

Available as LI 000 721

TABLE OF CONTENTS (continued)

TABLE OF CONTENTS (continued)

TABLE OF CONTENTS (continued)

Page

PART THREE

USER FEEDBACK PROCEDURES

Available as
ED 002 722

TABLE OF CONTENTS (continued)

TABLE OF CONTENTS (continued)

Page

PART FOUR

CLUSTERING METHODS

Available as
LI 002 723

TABLE OF CONTENTS (continued)

TABLE OF CONTENTS (continued)

TABLE OF CONTENTS (continued)

ERIC User Please Note:

This summary discusses all 5 parts of Information Storage and Retrieval (ISR-18), which is available in its entirety as LI 002 719. Only the papers from Part Five are reproduced here as LI 002 724. See LI 002 720 thru LI 002 723 for parts 1 - 4.

Summary

The present report is the eighteenth in a series describing research in automatic information storage and retrieval conducted by the Department of Computer Science at Cornell University. The report covering work carried out by the SMART project for approximately one year (summer 1969 to summer 1970) is separated into five parts: automatic content analysis (Sections I to IV), automatic dictionary construction (Sections V to VII), user feedback procedures (Sections VIII to XI), document and query clustering methods (Sections XII and XIII), and SMART systems design for on-line operations (Sections XIV and XV).

Most recipients of SMART project reports will experience a gap in the series of scientific reports received to date. Report ISR-17, consisting of a master's thesis by Thomas Brauen entitled "Document Vector Modification in On-line Information Retrieval Systems" was prepared for limited distribution during the fall of 1969. Report ISR-17 is available from the National Technical Information Service in Springfield, Virginia 22151, under order number PB 186-135.

The SMART system continues to operate in a batch processing mode on the IBM 360 model 65 system at Cornell University. The standard processing mode is eventually to be replaced by an on-line system using time-shared console devices for input and output. The overall design for such an on-line version of SMART has been completed, and is described in Section XIV of the present report. While awaiting the time-sharing implementation of the system, new retrieval experiments have been performed using larger document collections within the existing system. Attempts to compare the performance

of several collections of different sizes must take into account the collection "generality". A study of this problem is made in Section II of the present report. Of special interest may also be the new procedures for the automatic recognition of "common" words in English texts (Section VI), and the automatic construction of thesauruses and dictionaries for use in an automatic language analysis system (Section VII). Finally, a new inexpensive method of document classification and term grouping is described and evaluated in Section XII of the present report.

Sections I to IV cover experiments in automatic content analysis and automatic indexing. Section I by S. F. Weiss contains the results of experiments, using statistical and syntactic procedures for the automatic recognition of phrases in written texts. It is shown once again that because of the relative heterogeneity of most document collections, and the sparseness of the document space, phrases are not normally needed for content identification.

In Section II by G. Salton, the "generality" problem is examined which arises when two or more distinct collections are compared in a retrieval environment. It is shown that proportionately fewer nonrelevant items tend to be retrieved when larger collections (of low generality) are used, than when small, high generality collections serve for evaluation purposes. The systems viewpoint thus normally favors the larger, low generality output, whereas the user viewpoint prefers the performance of the smaller collection.

The effectiveness of bibliographic citations for content analysis purposes is examined in Section III by G. Salton. It is shown that in some situations when the citation space is reasonably dense, the use of citations attached to documents is even more effective than the use of standard keywords or descriptors. In any case, citations should be added to the normal descriptors whenever they happen to be available.

In the last section of Part 1, certain template analysis methods are applied to the automatic resolution of ambiguous constructions (Section IV by S. F. Weiss). It is shown that a set of contextual rules can be constructed by a semi-automatic learning process, which will eventually lead to an automatic recognition of over ninety percent of the existing textual ambiguities.

Part 2, consisting of Sections V, VI and VII covers procedures for the automatic construction of dictionaries and thesauruses useful in text analysis systems. In Section V by D. Bergmark it is shown that word stem methods using large common word lists are more effective in an information retrieval environment that some manually constructed thesauruses, even though the latter also include synonym recognition facilities.

A new model for the automatic determination of "common" words (which are not to be used for content identification) is proposed and evaluated in Section VI by K. Bonwit and J. Aste-Tonsmann. The resulting process can be incorporated into fully automatic dictionary construction systems. The complete thesaurus construction problem is reviewed in Section VII by G. Salton, and the effectiveness of a variety of automatic dictionaries is evaluated.

Part 3, consisting of Sections VIII through XI, deals with a number of refinements of the normal relevance feedback process which has been examined in a number of previous reports in this series. In Section VIII by T. P. Baker, a query splitting process is evaluated in which input queries are split into two or more parts during feedback whenever the relevant documents identified by the user are separated by one or more nonrelevant ones.

The effectiveness of relevance feedback techniques in an environment of variable generality is examined in Section IX by B. Capps and M. Yin. It is shown that some of the feedback techniques are equally applicable to collections of small and large generality. Techniques of negative feedback (when no relevant items are identified by the users, but only nonrelevant ones) are considered in Section X by M. Kerchner. It is shown that a number of selective negative techniques, in which only certain specific concepts are actually modified during the feedback process, bring good improvements in retrieval effectiveness over the standard nonselective methods.

Finally, a new feedback methodology in which a number of documents jointly identified as relevant to earlier queries are used as a set for relevance feedback purposes is proposed and evaluated in Section XI by L. Paavola.

Two new clustering techniques are examined in Part 3 of this report, consisting of Sections XII and XIII. A controlled, inexpensive, single-pass clustering algorithm is described and evaluated in Section XII by D. B. Johnson and J. M. Lafuente. In this clustering method, each document is examined only once, and the procedure is shown to be equivalent in certain circumstances to other more demanding clustering procedures.

The query clustering process, in which query groups are used to define the information search strategy is studied in Section XIII by S. Worona. A variety of parameter values is evaluated in a retrieval environment to be used for cluster generation, centroid definition, and final search strategy.

The last part, number five, consisting of Sections XIV and XV, covers the design of on-line information retrieval systems. A new SMART system design for on-line use is proposed in Section XIV by D. and R. Williamson, based on the concepts of pseudo-batching and the interaction of a cycling program with a console monitor. The user interface and conversational facilities are also described.

A template analysis technique is used in Section XV by S. F. Weiss for the implementation of conversational retrieval systems used in a time-sharing environment. The effectiveness of the method is discussed, as well as its implementation in a retrieval situation.

Additional automatic content analysis and search procedures used with the SMART system are described in several previous reports in this series, including notably reports ISR-11 to ISR-16 published between 1966 and 1969. These reports are all available from the National Technical Information Service in Springfield, Virginia.

G. Salton

XIV. A Prototype On-Line Document Retrieval System

D. Williamson and R. Williamson

Abstract

A design is outlined for a SMART on-line document retrieval system, using console initiated search and retrieval procedures. The conversational system is described as well as the program organization.

1. Introduction

The SMART system presently contains routines for experimental, off-line document retrieval. The experimental results obtained so far indicate that automatic document retrieval can provide useful information for general library users. The next logical step is the development of a suitable user-oriented interface providing access via on-line consoles in an interactive manner.

This report describes a prototype, on-line document retrieval system and a user interface. The system which is outlined is intended to provide the best service possible to on-line users at a reasonable cost, but could also be efficiently used with very few modifications as a batch or remote entry system. While initial tests with collections of only a few thousand documents and less than five consoles is anticipated, the mechanisms used are intended to be applicable without revision to much larger collections of about 500,000 documents, and up to one to two hundred input-output consoles.

2. Anticipated Computer Configuration

In order to provide adequate response times — about 10 seconds for minor inputs and about 30 seconds for responses to search commands — a large, high-speed computer is necessary. Document retrieval, like many other non-numeric processes, requires a large data base of which a small, but substantial, fraction must be accessed for each query. Thus, it is necessary to operate with large, on-line files — presumably on a disk (although certain files could be placed on a data cell type device).

While a large computer is necessary to support the input-output equipment, and provide reasonable response times, an on-line retrieval system such as SMART, will not be able to utilize the full resources of a large machine. First, periods will occur when no users wish to avail themselves of the on-line system; and even when actual users are present, most of the real-time of an interaction is spent waiting for user decisions. Also, while processing a search request, the computer may be expected to be input-output (I-O) bound waiting for vocabularies and documents to be brought into core.

If processing costs are to be reasonable, provision must be made to permit non-retrieval users to process while the retrieval system is inactive for one reason or another. The type of environment needed is typified by many of the multi-processing and time-sharing systems available on large machines today. With these systems, jobs are effectively allocated to two queues: most are awaiting execution, and a few are in execution. Those in execution share the central processor (C.P.U.), memory, and on-line storage devices. Each memory area and storage device is usually dedicated to a single job. (In addition, a few devices and storage areas are normally reserved for the supervisor which is used by all jobs.) CPU allocation is normally switched from one <u>executing</u> job to another (through the supervisor) whenever that job is blocked — usually because it must await completion of an I-O transmission. System blocks are provided to prevent jobs from monopolizing the CPU, when no blocks occur for a certain time.

In the normal course of events, each executing job receives the opportunity to use the CPU several times a minute. Much of the time, a retrieval process such as SMART will be unable to utilize the opportunity to process. However when SMART has work to do, and the information necessary to do that work is available, the CPU is normally accessible — effectively instantaneously. The reason is that the retrieval tasks will appear as highly I-O bound jobs, which are therefore core resident for long periods of time, and are usually high in priority for CPU access.

SMART can make efficient use of as much core storage as can be made available. However, the retrieval routines tend to be small, and are highly overlayable; thus, the basic core area requirements are quite small. As in other typical data processing applications, the major core requirements in a retrieval program are for data areas in which to place I-O buffers for dictionaries, documents, etc. It would be most desirable if SMART could obtain 100 K to 200 K of core (possibly from a bulk core rather than from the high speed main core) on demand, for periods of only several seconds each time a request (or group of pseudo-batched requests) are processed. This core could easily come from the system buffer pool. However, sharing of core in this way is not a normal feature of today's operating systems; thus, SMART will undoubtedly have to reserve an area of high-speed core for programs (25-30 K bytes), and an area of bulk core for data (at least 50 K bytes — however, the more core is available, the faster will normally be the obtainable response times).

3. On-Line Document Retrieval — A User's View

When control of a console is transferred to SMART, the remote unit should be titled clearly to indicate to the user what basic information is needed at each step (detailed information should be provided as specified by a user's manual).

If SMART is on-line at the time of console transfer, the user must first enter such basic information as his name and account number (see Fig. 1). After this information is accepted by SMART, the user can proceed to ask for the execution of a given process. Many processes, such as query searches, query updates, and displays of output are available.

An initial user will probably start with a single query search (such as shown in Fig. 1). In this case, he will type in his query and then ask for a search to be done. The results will be displayed (in one of several possible forms, such as titles, abstracts, etc.), and the user will then either get a further display of the documents, or use the results of the search at that point.

Several types of display for retrieved documents could be used. The volume of information included in abstracts (or full articles) is likely to be so large that teletype display will be impractically slow; cathode-ray tube display is however quite expensive. Storage of abstracts at the remote terminal is an attractive alternative, with storage either on microfiche cards or in computer listings.

Following the retrieval of an initial set of abstracts, the query

```
$PROCEED
-SMART
#
#         SMART
#
#What is your name?-
-Joe Cornell
#What is your access code?-
-NONE
#Your access code is "MNAIZ".
#Do you wish to enter a query?
-Yes.
#Please enter your 1 th query.
#Type "End of query." when finished.
-What articles are there in ...
-
-                       . End of query.
#Is your query ready for analysis?-
-Yes.
```

A Typical User's First Query

Fig. 1

Rank	Article	Correlation
1	60x1212 L. B. Heilprin, Towards a Definition of Information Science	0.6708
2	45x1215 D. Crosland, Graduate Training in Information Science	0.4472
3	03x1210 R. L. Taylor, In Information Science Education	0.3823
4	21x1209 Personnel – An Assessment and Projection	0.3660
5	43x1206 A. M. Rees, The Education of Science Information Personnel – A Challenge to the Library Schools	0.3651

Results of Initial Search of Query 1

Fig. 1 (continued)

Following the retrieval of an initial set of abstracts, the query author can return to the console and give the system his estimated relevance decisions. Since a prime source of error in all document retrieval systems is the discrepancy between a query author's intended query and his expressed query, initial queries can often be greatly improved through a process known as relevance feedback. This process modifies the query by adding words used in the relevant documents to the query, thus enlarging, and hopefully, improving the query. To improve his query, the user would re-enter the system, asking for a search on the original query plus relevant documents. An example of the re-entry to use feedback is shown in Fig. 2. In this case, the user asks to delete titles and uses only minimal replies. After the preliminary sign on at the console, the user is asked if he wishes to submit relevancy decisions for any active queries (in this case query 10). An indication must then be given of these decisions on a relevance scale from 1 to 5. After entering the decisions, the user asks for relevance feedback, and gets the results in a manner similar to the search results in Fig. 1.

For more experienced users, other procedures might be useful. Dictionary display to help the user construct more reasonable queries is possible, and various types of syntactic analysis can be used. The user can also alter the searching methods used by utilizing his private search parameters instead of the standard system parameters.

Each of the various procedures available to users requires specific patterns of interaction between the console and the user. Table 1 contains a tabular display of portions of a proposed console interface. Only a few of the procedures are traced in full, as an example of how such an interface would be constructed. The importance of the table lies in its overall structure — the specific wording of the messages and the division of labor among table segments is of minor interest. However, it should be noted that console interaction is handled in a sequential manner. Thus each user is associated with just one pointer indicating the segment to which he is replying.

Each table segment consists of one computer to console message including a possible user response, or system action. If an unanticipated response is obtained in a basic system, the text will be repeated in tutorial mode. In a more advanced system, special segments could be set up to handle unanticipated responses in special ways.

Several responses are global in that they could appear at any time rather than in response to a specific SMART message. These are listed in Table 1 under segment 0 (e.g. reply class shifts). The normal form of a response is a key phrase followed by a carriage return. Some responses can include explicit requests for changes in parameter values at the user's option. For those responses which can take up more than one line, a period terminates the response.

Some responses can contain a number of periods, and consist of more than one line, e. g. queries. Such responses are terminated by a key phrase, e.g. "End of query.". To eliminate problems caused by missing periods, etc., a user should be required to enter at least one character within 10 seconds of a carriage return; otherwise the multiple line response is considered complete. Such a rule is needed to prevent the system from waiting for user action while at the same time the user is expecting action by the computer.

Each reply text uses an ampersand "#" to indicate a mandatory carriage return. Additional carriage returns are inserted as needed by a console message director depending on the number of characters per line available on a

```
$ Proceed
- SMART.
- No title, minimal replies.
# Name?
- Mike Lesk
# Access Code
- XAQ13
#
#            SMART                XAQ13   Mike Lesk
# Relevancy decisions for active query 10?
- Yes.
# Document # 405  603  201  815  10004
- Decisions  3,4,3,5,1
# Abstract decisions?
- Yes.
# Relevance Feedback?
- Yes.
# Search?
- Yes, search.
#
#     Results of 3rd search of Query 10
.
.
.
- DONE
# Control is relinquished to the supervisor.
$ Proceed
```

Relevance Feedback

Fig. 2

specific console. A hyphen "-" indicates that the console keyboard is unlocked for a user response. Each quoted anticipated response, such as the key phrase responses, can be abbreviated by using only the capital letters specified in the response. All anticipated responses can be typed using any mixture of upper or lower case letters.

The contents of the 'Internal' column are, for the most part, self-explanatory. The use of the variable READY is described later but included in the Table for completeness. It indicates whether console interaction is needed, or whether internal work is needed.

The 'Next Segment' field indicates which segment is to be considered next. Often this is dependent on the response or the Action field. An "R" indicates a return to whichever segment was previously considered. Each user is assigned variables to indicate the segment he is in and the line of text (for that segment's message) that is being transmitted. When a console joins SMART, logical control is first set at segment 9 if SMART is on-line, otherwise control is set at segment 1. Note that segments above 104 are not included in the Table, but would be set up in the same way as other segments.

4. Console Driven Document Retrieval — An Internal View

This section describes a possible implementation of the on-line document retrieval system presented earlier. All routines available for batch SMART runs are usable without any reprogramming. An on-line executive program is however needed to drive the consoles and the batch routines.

A) The Internal Structure

The internal structure needed for a prototype system must satisfy several goals. As indicated in the introduction, a prototype system must

Segment Number	Reply Class	Messages for Consoles	Anticipated Responses from Consoles	Internal Action	Next Segment
0		(none)	"DONE"		51
			(Attention Key)	Delete transmission and activate keyboard	R
			"Tutorial Replies"	REPCLS = Tutorial	R
			"Short Replies"	REPCLS = Short	R
			"Minimal Replies"	REPCLS = Minimal	R
			"?" or an unanticipated response	If REPCLS = M Then REPCLS = S	R
				If REPCLS = S Then REPCLS = T	R
				If REPCLS = T	9000
1				SMART is on-line	2
				SMART is not on-line	3
2		#SMART is already on-line. You may not initiate a duplicate system.			51
3		#SMART is initiated. Your console is the master console. May other consoles attach to SMART?-	"Yes"	NEWCON = Yes	3.5
			"No"	NEWCON = No	3.5
3.5			(Reply Class Shift Only)		4

a) Introductory Segments

SMART Console Interface

Table 1

Segment Number	Reply Class	Message for Consoles	Anticipated Responses from Consoles	Internal Action	Next Segment
4	S M	#What is your name?- #Name?-	User's Name	Store Name	6
6	S M	#What is your access code? #Access code?-	"None" Access code	Assign an access code Verify code-OK NOK	 7 8 9900
7		Your access code is "ACCODE".		NUMCUS(-number of customers ACCODE(-User's new access code Store access code	 100
8		#Welcome to SMART. <u>ACCODE</u> <u>NAME</u>		ACCODE(-access code NAME(-User's name as on file Does user have any unfinished queries? Yes No	 2000 100
9				If SMART is on-line If SMART is off-line	3.5 10
10		#SMART is not now on-line. Retrieval will be available (time, day).			51

a) Introductory Segments (contd.)

SMART Console Interface

Table 1 (continued)

Segment Number	Reply Class	Message for Consoles	Anticipated Responses from Consoles	Internal Action	Next Segment
50	S	#Please select one of the following programs...	"Done."		51
			"Query."		100
			"Analyze."		500
		#Query, Analyze, Search, Display, Feedback, Pre-search, Search Options, Feedback Options, Analysis Options, Judgments, Done.	"Analyze using XYZ strategy."	ANALPV=XYZ	500
			"Search."		1000
			"Search, using XYZ strategy."	SEARPV=XYZ	1000
			"Display."		2000
			"Feedback."		3500
			"Feedback, using XYZ strategy	FEEDPV = XYZ	3500
			"Judgments."		3000
			"Pre-search."		4000
			"Analysis options."		5000
			"Search options."		6000
			"Feedback options."		7000
51		#Thank you for using SMART #Control is relinquished.		READY = 0 TST = 0 Return control of console to supervisor	

b) Central Director

SMART Console Interface

Table 1 (continued)

Segment Number	Reply Class	Message for Consoles	Anticipated Responses from Consoles	Internal Action	Next Segment
100		Do you wish to enter a query?	"Yes."		101
			"No."		50
101	S	#Please enter your MAXQUEth query. #Type "End of query." when finished.		MAXQUE = MAXQUE + 1	
				NUMQUE = MAXQUE	102
	M	#Enter MAXQUEth query.			
102		-	A line of a query.	Store line.	
				Does line end in EOQ? YES	103
				No	102
103	S	#Is your query ready for analysis?-	"Delete Query."	MAXQUE = MAXQUE - 1	
				Delete query	101
	M	#Analyze?-	"Add to Query."		
			"Boolean."	Does user want to supply Boolean Informa-tion? YES	104
				No	500
			"Yes, Search."	DOANAL = 1	
			"Yes."	DOCENT = 1	
				DOSEAR = 1	500
			"Yes, Search, using XYZ Strategy."	As above and SEARPV = XYZ	500
			"Yes, using XYZ Strategy."	DOANAL = 1	
				ANALPV = XYZ	
			"No."		50

c) Query Text Handling

SMART Console Interface

Table 1 (continued)

have the speed and ease of use of a production system, as well as the flexibility and measurability of an experimental system. A document retrieval system must provide fast on-line service and exhaustive, inexpensive off-line service. A typical first thought is simply to provide two systems — one for on-line work, and the other for off-line work. However, a single, flexible system capable of handling both types of service is normally less expensive to develop, operate and maintain than two separate systems, provided a scheme with the needed features can be found.

The flexibility required to provide on-line and off-line service in a single package is best illustrated by the differing amounts of transmitted information. Off-line users will want, and can afford, to use large volumes of information. Such a volume of information cannot be transmitted at low cost to an on-line user, nor would an on-line user be able to cope with the quantity of information of use and interest to an off-line user.

Another illustration of the needed flexibility is related to machine storage. During off-hours, ownership of large amounts of storage for long lengths of time may be possible. Most on-line requests, however, will be serviced during the day when others also want to use the computer. To reduce costs, it is necessary that a minimum of computer resources be permanently allocated to each specific task. Unfortunately, human response times are much slower than normal computer response times when the computer is being used for batch processing. For example, a complete off-line search for 42 queries and 1400 documents can be completed in less real-time than a single on-line query because of the slowness of human response. (Obviously, the 42 query search requires more process time.)

B) General Characteristics of SMART Routines

To satisfy the need for flexibility and modifiability, SMART is programmed as a set of small, clearly defined, and well documented Fortran subroutines. Each subroutine accomplishes one task with a minimal interface with other routines. Each SMART routine lies in a distinct class depending on the amount of structure in the data used or manipulated. On the bottom of the pyramid are the I-O routines and the MOVE routines (which move sets of sequential locations from one place to another). These routines "know" only the length and origin of the fields with which they deal.

Next in the hierarchy are routines which deal with the various kind of vectors. SMART uses several kinds of vectors, all consisting of a "head" indicating the length of the vector followed by information in double words. In the case of concept vectors, these double words contain concepts and weights; in the case of result vectors, the first word contains the document number and rank retrieved (each in half words), and the correlation of the document with the query. The routines that deal with these vectors "know" the internal structure of the vectors. Some examples of this class of routine are LSTCON, which prints the contents of a concept vector, and RESULT, which prints the contents of a vector of document-query correlations.

Above this level are routines which deal with groups of vectors. These are the routines which know that many queries exist in the system. Typical of these routines is BLOCK, which combines the result vectors for the several iterations of one query during a batch run, and gives the combination, one query at a time, to RESULT.

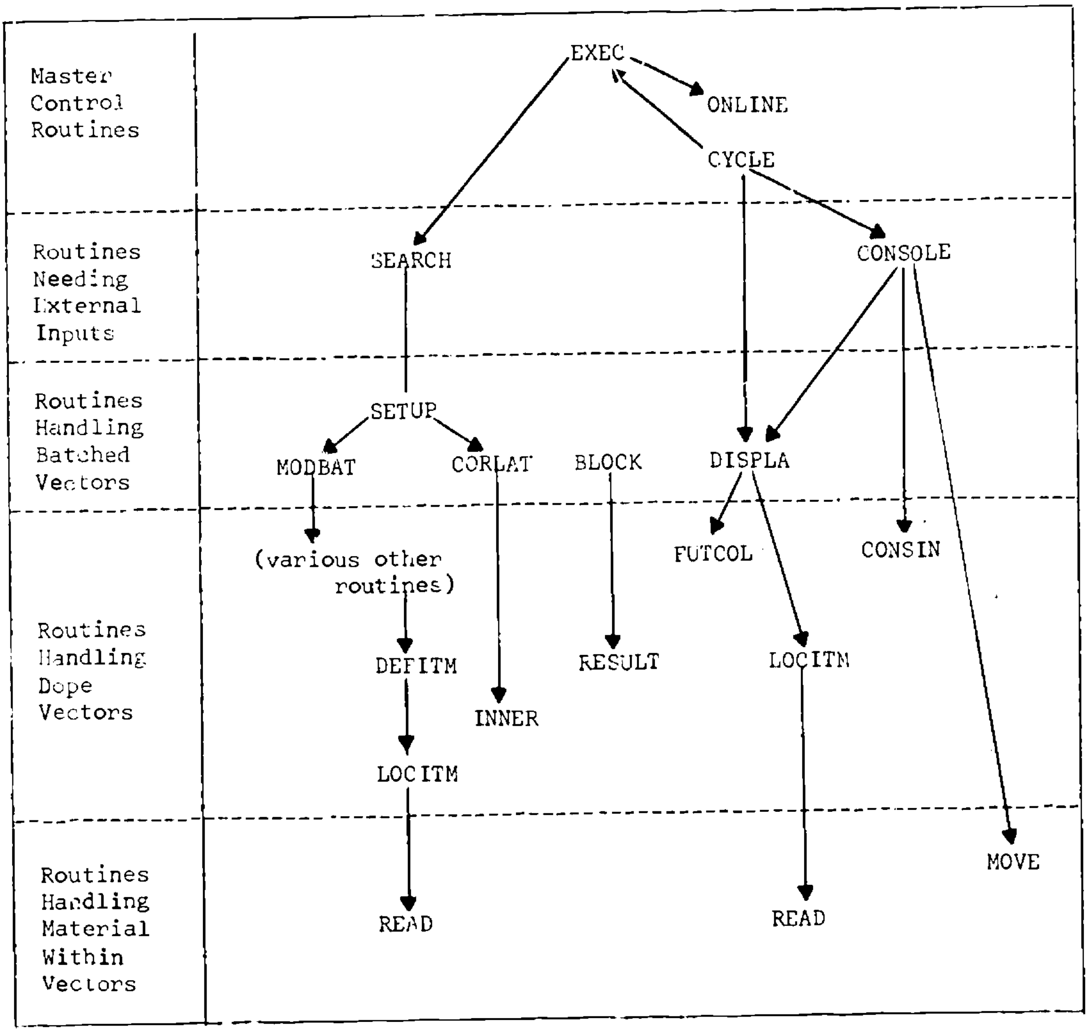
At the top to the entire pyramid are the routines EXEC and ONLINE. EXEC is a card-controlled driver for the system. It is normally used for batch experimental work and jobs typically done off-line, such as the addition of new text and centroid generation. ONLINE is normally used to control on-line document retrieval. A partial tree of SMART routines showing this structure follows in Fig. 3.

C) Pseudo-Batching

Basic to an understanding of the mechanism proposed for document retrieval is the idea of pseudo-batching. In any reasonable batch-processing document retrieval system, a large number of queries are handled in parallel. This serves to reduce the fixed overhead per query to a fraction of the total overhead. So long as the increased expense of dealing with several queries is kept small, there is a net gain in effectiveness per unit cost.

A basic problem in an on-line document retrieval system is that each search passes through different stages with different requirements. This presents problems because of the multiplicity of distinct programs which may be required, as well as the input-output problems. If each query is multi-programmed with other queries, severe competition for resources would result. One query would need document files, another dictionaries, and yet another would require text files. A complicated scheduling algorithm would be required to untangle the requirements for file access facilities and storage space; this would increase overhead costs sharply.

In an on-line system where many users individually cycle through the same set of routines and files, a much better utilization of resources results by batching the incoming queries. If the system processes only



Structure of SMART Routines

Fig. 3

those queries available at the start of a (twenty second) cycle, competition for resources is eliminated. Each query would then take thirty seconds on the average; twenty seconds of actual processing and ten seconds of waiting.

Many advantages can be accrued to the overall system and thus to the user by the batching of queries. Of greatest importance is the resulting lack of competition for different files or for space to store them. Secondly, each query has an apparent overhead considerably less than it would have if it were the only query to use a file at a given time. Obviously, lower overhead means lower cost.

D) Attaching Consoles to SMART

Since one can assume that consoles will not be continuously dedicated to a document retrieval system, at least in an experimental environment, provision must be made for transfer of control of a console from the computer supervisor to SMART. If SMART is core-resident and a specific console is wanted for SMART, the process is as simple as obtaining additional disk space or more core. However, it is desirable that a user be able to go to any available, supervisor-controlled console, and that the console be transferred to SMART at the user's initiation. Under such circumstances, the possibility also exists that SMART is not available on-line at some given time. Naturally, the problems and cost of serving additional users are far less when SMART is already on-line than when SMART must be started for the first user. Since SMART wishes to permit anyone to utilize the document retrieval system, provision must be made to prevent the occurrence of unreasonable expenses. One obviously unreasonable expense is the improper activation of SMART. Another problem is the need to keep to a minimum the actions which the typical, non-computer-oriented user must carry out to use the SMART system on-line.

For these reasons SMART could include a small routine that is continuously a part of the supervisor. Normally, after a user has activated a console (e.g. by dialing the computer if telephone lines are used), the computer expects the name and account number of the user (in order to prevent unauthorized usage). The user may then enter simply the word "SMART". This will cause the execution of a program called SMTLATCH which is supplied with the "name" of the console presently wanting SMART. This code will "know" whether SMART is on-line or not.

If SMART is not on-line an appropriate response is made. (An example is presented in Fig. 4.) If SMART is on-line, the console number of the new user will be made available to the normal SMART programs and a flag will be set indicating that a new console needs to be attached. When SMART regains use of the computer, the supervisor can be requested to transfer control of that console to SMART.

```
(Dial computer and press carriage return.)
#Proceed.
%SMART.
$SMART will be available next at 3 p.m.
   Tuesday, October 4, 1968.
#Proceed.
%
```

Console Response to a Request for SMART
When SMART is not On-line

Fig. 4

E) Console Handling — The Supervisor Interface

SMART will not need to worry about physical control of the consoles. Rather SMART provides a routine which the supervisor can call whenever a new line is available from a console. The console keyboard is then locked (i.e. nothing more can be typed by the user) until SMART allocates space for a new line somewhere in a SMART section of memory and so tells the supervisor. Alternatively, at this time, SMART can transmit a line to the console. Normally the console keyboard will be freed fast enough (if multi-line input is anticipated) so that the user will be unaware of the keyboard ever being locked.

When SMART wishes to write on a console (which includes unlocking the console keyboard), a call to the supervisor is made with the location of a message and the name of specific console on which the message is to appear. If the keyboard of that console is locked, the message is immediately transmitted. If the keyboard is not locked, the transmission is refused and SMART will have to lock the keyboard first and accept whatever message was transmitted. (On the equipment presently available the console cannot be locked; only the user can lock the keyboard by pressing "Attention" or "Carriage Return"; the system must therefore wait for user action.)

F) Parameter Vectors

As each enquirer is introduced to SMART, he is associated with a user vector that contains pointers to parameter vectors. These vectors are filled with information taken from control cards during a batch processing run, or from a default vector for new on-line users, or from personal parameter vectors. These parameters supply values needed to control the action of the retrieval routines. Each user may define his own personal parameter vectors which can be saved for use on many searches.

G) The Flow of Control

The flow of batched queries is comparatively simple ~~compared~~ to that of on-line queries. Although batched and on-line queries use different means to fill parameter vectors, and take different action with respect to the output of most routines, these differences are unimportant.

The manner of introducing an on-line user has already been described. (As far as SMART is concerned, a user and the console he is then using are equivalent in all ways. Thus, wherever the word 'console' appears, the word 'user' could be substituted.)

The on-line control program consists of two logically distinct routines. CONSOL handles physical communications with the consoles on an interrupt basis (i.e. in real-time). CYCLE handles the use of core and the large system files by cycling among them, satisfying users as it can. Logical control of each console shifts between CONSOL and CYCLE.

The SMART On-Line Console Control Block (SOCCB) indicates at any given instant which routine is logically in command of a console. The SOCCB synchronizes the real-time routine CONSOL with the process-time routine CYCLE. The READY flag associated with each console takes on certain values if the console is awaiting completion of a task done by CYCLE. When CYCLE is finished, the READY key is changed. Since the key is changed, CONSOL can recognize that it should proceed with that console.

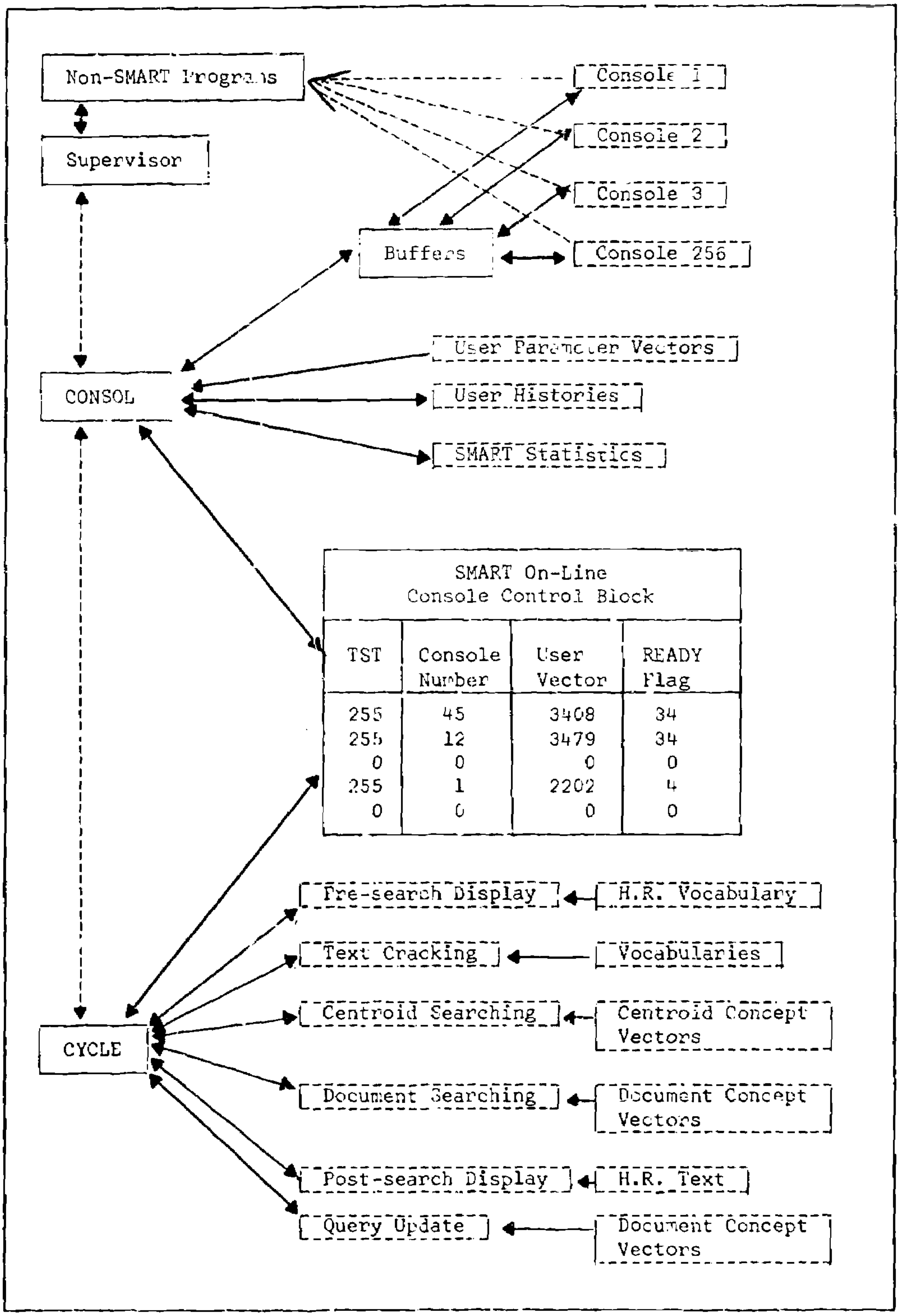
Testing READY flags (for up to 256 consoles) is accomplished by a single instruction (Translate and TEST — TRT) using a 256 byte array. Since the test is fast, it can be carried out frequently by both CONSOL and CYCLE. For example, after sending each line of a message to one console, CONSOL can test to see if any other console requires service for a single line. If so, the servicing of the one console with a series of lines is terminated, and consoles with single-line needs are handled. CONSOL then returns to the multi-line message and finishes. CYCLE uses the speed of the TRT instruction to locate those queries needing a specific process. After each CYCLE driven routine finishes with a batch of queries, the table can be scanned to see if, in the meantime, any other queries now need that same process. Some routines which can be logically divided into two parts, one essentially in-core and the other necessitating file accessing, could be programmed to check for "latecomers" to speed up overall response without losing the advantages of cycling.

For a list of typical READY flags see Table 2.

H) Timing Considerations

In order for the type of organization presented to be acceptable to non-SMART users of the computer, two timing considerations are paramount. First the CONSOL routine must be assigned highest priority by the supervisor, since it must respond to on-line signals. CYCLE is assigned the second highest priority. This implies that if CYCLE is free to perform work, the CPU is taken away from any other executing program (except CONSOL and the supervisor itself). Normally, however, CYCLE is I-O bound. While CYCLE is waiting for needed information from noncore resident files, and when CYCLE has no work to do, the CPU is able to do the work of other customers.

Thus, CONSOL must have available everything it needs to work and CYCLE must contain <u>no</u> wait loops of any size. If information is not available, the supervisor must be given control until the required information is available.



Legend: Core-resident ———— Auxiliary ----- H.R. Human Readable

SMART On-line Control Logic

Fig. 5

Routine Needed	READY	Meaning
(NONE)	0	Unused slot.
CONSOL	1	Newly arrived console, no assigned user vector.
(NONE)	2	Console keyboard unlocked for user transmission.
CONSOL	3	Console keyboard locked by receipt of a user transmission.
(NONE)	4	One line message going to console.
CONSOL	5	Console keyboard locked further lines are needed.
CYCLE	6	Allocate core.
CONSOL	7	Core Allocated.
.	.	.
.	.	.
.	.	.
CYCLE	20	Crack text.
CYCLE	21	Cracking text.
CONSOL	22	Text cracked.
CONSOL	23	Notifying user.
CYCLE	24	Set-up pre-search display.
CYCLE	25	Setting-up pre-search display.
CONSOL	26	Pre-search display setup.
CONSOL	27	Displaying to user.
CYCLE	40	Search centroid tree.
CYCLE	41	Searching centroid tree.
CONSOL	42	Centroid tree searched.
CONSOL	43	Informing user of results of tree search.
.	.	.
.	.	.
.	.	.

READY Flags

Table 2

I) Noncore Resident Files

Before going into CONSOL and CYCLE in detail each of the files used by SMART is introduced briefly. The various logical segments of core are then similarly defined to provide a reference and to eliminate detailed descriptions within succeeding sections.

SMART files can be divided into three distinct classes — those used by CYCLE, those used by CONSOL, and the consoles themselves. The console files are basically standard sequential files, with, however, an unpredictable access time. Like sequential files, records are read (or written) one-at-a-time and in linear order. There is, of course, no backspacing, rereading or overwriting.

CONSOL deals with three files of a more familiar nature. The 'SMART Statistics File' is a sequential, write-only file on which is placed information to enable evaluation of SMART's performance by supervisory staff. Information such as observed user and SMART response times, and statistics on query authors using the system might be kept.

The 'User History File' retains information about unfinished queries on an individual user basis. For each user, such information as the number of queries he has submitted to the system, the number still active, and accounting information may be kept. For each active query, a record is kept of the text of the original query, and of the last active concept vector for that query. Perhaps, a list of additional documents, unseen by the user, should be kept to try to forestall a complete lack of positive feedback. In this manner costs could be kept reasonably low for a majority of users by not showing many documents except when necessary. One might also want to keep some type of record of the searched centroid tree so that "obviously" unsuitable tree nodes would not have to be reconsidered during relevance feedback.

The 'User Parameter Vector File' contains user parameter vectors. Each user can have several different parameter vectors (with distinct names) for different purposes. The only reason for separating this file from the previous file is that this file is essentially a read-only file, whereas the previous file is updated with every system access. It is anticipated that the directory for this file would be one part of the preceding file.

The files used by CYCLE-called routines are of two distinct types — human readable and machine readable. The human readable files contain information suitable for display to normal users at consoles. The other files are however organized for maximum speed of access and minimum space for storage of information used solely by SMART. A complete system must have human readable files -- the vocabulary aid files and the source text files. Vocabulary aid files contain thesaurus expansions, hierarchies, frequency lists, etc. Source texts contain titles and abstracts of documents in a form suitable for on-line display. Normally vocabulary aids are used prior to a search and texts after a search.

There are three machine-readable classes of files — vocabulary files, files of centroid concept vectors, and files of document concept vectors. Vocabulary files contain the information needed to quickly understand input text (i.e. to convert raw text into a standard concept vector). The other two files contain, respectively, files of centroid concept vectors and files of document concept vectors. The separation of centroid and document concept vectors into two distinct files is dictated by the relative sizes of the two files. Commonly, a centroid has over 10 sons: thus a centroid tree for a file of 100,000 document would contain less than

9,000 nodes. In most situations, the centroids could be accessed faster as a separate data set because of their smaller volume.

There also exists a file which contains the programs called by CYCLE. In order to further conserve space, it may be desirable that these mutually exclusive routines be overlayed during execution.

J) Core Resident Files

Seven types of core resident files are used by SMART. They have differing typical lifetimes, lengths, sources, and destinations. Because of their differing lifetimes, they are allocated from different pools of available core. This minimizes a serious tendency to fragment core and eliminates a need for dynamic relocation of in-core files. By permitting the system to obtain variable amounts of core, SMART is able to work in 50 K or 500 K, albeit with grossly different response times and CPU utilization rates.

The first file is the previously mentioned SMART On-line Console Control Block (SOCCB). This block is the key to the entire control of the on-line system and is, therefore, described in detail in the next section. The size of the SOCCB is fixed when SMART is initiated by the number of consoles to be accepted on-line at one time. This block is retained until SMART goes off-line.

Each user is assigned a user vector. This block is of fixed length and is retained as long as the user is on-line. The user vector contains pointers to the locations of dynamic fields "owned" by the given console. These fields include parameter vectors, buffers and correlation vectors. The user vector is accessed only by CONSOL and CYCLE.

The parameter vectors contain values for variables used to control the various routines. Each routine needs its own parameter vector. There exists a standard default parameter for every routine, and these standard vectors are core-resident for the life of a given invocation of SMART. Any user vector can point to one of these default vectors; however, no user can change values in the default vectors. If a user wishes to change any values, space is allocated for his own individual parameter vector for each routine the user wishes to control in a non-standard fashion. A user may name his parameter vectors in order to re-use them easily. An individual parameter vector is core-resident only for the duration of the process which that vector controls.

Buffers contain a line or a track of information. They typically have a short lifetime, and the space occupied by the buffers is reutilized at a high rate. Buffers to or from a single file can be linked while in-core. These vectors constitute the majority of core needed by SMART. In some cases, it may be desirable to keep some buffers in core in anticipation of repeated use. If sufficient core is available, this can be done. However, this in-core saving of a buffer is unknown to all routines except to the buffer manager. This permits a routine to use 50 K of 500 K bytes without any internal knowledge. Only the response times to requests for a buffer will differ depending on the amount of core utilized.

The concept vectors constitute the output of the routines converting text into concept vectors, and of the query update routines. These vectors are much shorter than the text they represent, and they can be more easily utilized for search purposes. Only one concept vector per user need be kept in core and the concept vector supplants the buffers containing the original query.

Specification and correlation vectors contain the names of individual centroids or of documents to be matched with a query, and later, the correlations with those items. The life of these vectors is short but the core requirements for a single query can be determined only dynamically.

Result vectors are shortened correlation vectors. They are used by CONSOL to pass information to the consoles.

5. CONSOL — A Detailed Look

Once the overall structure of the proposed on-line system is understood and the contents of the various files in understood, a detailed explanation of the operation of the two major routines becomes straightforward.

CONSOL will be considered first since it is first logically. Before going into the routine itself, the SMART on-line console control block (SOCCB) is described:

A) Competition for Core

It is possible that one user may finish a line and the interrupt-called supervisor can start CONSOL, while a second user can finish his line before CONSOL finishes with the first user. The second user's finish would cause the supervisor to start CONSOL again. A routine like CONSOL is called reentrant if several different processes (users) can simultaneously execute it. On a single CPU machine like Cornell's 360/65 the simultaneity is apparent and due to interrupts. However, on a multiple CPU machine the sumultaneity could be real. In both cases the problem is the same: no process can know if another process is also executing the same code. The requirement is that no "edition" of a reentrant routine can change core locations possibly known to another "edition" of that routine. If the reentrant routine must obtain additional core, the same problem exists — two editions may try to take the same space. A similar problem arises between CONSOL and CYCLE: CYCLE could be claiming an area of core while at the same time CONSOL decides to use that same area.

In order to prevent destructive competition for ownership of resources, the 360 provides a single instruction which locks a resource as it tests that resource for availability. The instruction is called Test and Set (TST). Basically TST sets a byte non-zero and sets the condition code to zero or non-zero as the previous contents of the byte were zero or non-zero in one inseparable step. (The TST instruction is outlined in Fig. 6).

B) The SMART On-line Console Control Block

The SMART On-line Console Control Block (SOCCB) shown in Fig. 5 holds four items for each active user. The maximum number of consoles that can be on-line at one time is decided when SMART is first entered; MAXUSERS contains this number. The fields marked TST and READY (in Fig. 5) are each vectors of "MAXUSERS" consecutive bytes. The TST field contains zero if that particular line is unused. When a line is reserved for a particular console, the TST field is set non-zero. The Console Number field contains the supervisor number for a console and the User Vector field contains the location of the user vector for that console.

TST LOCK (Instruction)		Before Execution	After Execution
Case 1	Location LOCK	0	255
	Condition Code	-	zero
Case 2	Location LOCK	255	255
	Condition Code	-	non-zero

The Test and Set Instruction (TST)
as Applied to the Location Named LOCK

Fig. 6

C) The READY Flag and the TRT Instruction

The READY field contains one of 256 equivalent flags. Each flag (value) indicates what process is then needed by that user. Typical values are given in Table 2. To understand the value of the vector, one needs to understand the Translate and Test (TRT) instruction. This instruction considers two read-only vectors. The first vector is the vector of READY values; the second contains a table of 256 bytes. This last table contains zero bytes except in those bytes whose address (relative to the first byte of the table) is the same as a READY value which must be tested. The TRT instruction takes bytes from the first vector, one-at-a-time, and looks at the table entry corresponding to the value of that byte. If the object byte is zero, the next READY value is considered; if the object byte is non-zero, the instruction ceases with that object byte and the location of the source byte is made available. If no byte stopped the instruction, that fact is so indicated. If the instruction is stopped by a non-zero object byte, the registers used by the instruction are left in a condition such that the instruction can be reexecuted for the remaining bytes in the source vector. A pictorial explanation of the TRT instruction is given in Appendix 1.

For internal convenience, READY values are often assigned in blocks — each block associated with a given process. Most processes can be divided into four phases: unconsidered by CYCLE, being considered by CYCLE, unconsidered by CONSOL, and being considered by CONSOL. Some READY values appearing in Table 2 show this assignment.

D) The Routines LATCH, CONSIN, and CONSOT

When a person types "SMART" on a console, the supervisor transfers control to SMTLATCH. SMTLATCH interrogates the variable SMTOPEN. If SMTOPEN is zero, SMTMSG (containing the appropriate message) is sent out to the calling console. If SMTOPEN is non-zero, control is transferred to (the location contained in) SMTOPEN. SMTLATCH, including SMTOPEN and SMTMSG, is always available to the supervisor as a standard supervisor process. Since SMTLATCH takes only 96 bytes, it can be kept constantly core-resident.

The first routine called when SMART is started in the standard manner is (ONLINE) which inserts the location of the routine LATCH at SMTOPEN. When SMART no longer wishes to accommodate new users, the routine OFFLINE updates SMTMSG to indicate the next scheduled time for on-line document retrieval; finally, SMTOPEN is set to zero. Consoles active in the system can still be accommodated in any suitable manner.

When LATCH is called, an unused row is located in the SMART On-line Console Control Block (SOCCB) using the TST to insure that the selected row is indeed available. LATCH then changes READY for that row to 1 (from 0) and stores the name of the console in the SOCCB. If CONSOL is running, LATCH simply returns to the supervisor (which will restart CONSOL where CONSOL was interrupted). If CONSOL is not running, LATCH causes the supervisor to mark CONSOL as runnable. LATCH then returns to the supervisor. The new console will be noted in due course by a TRT in CONSOL.

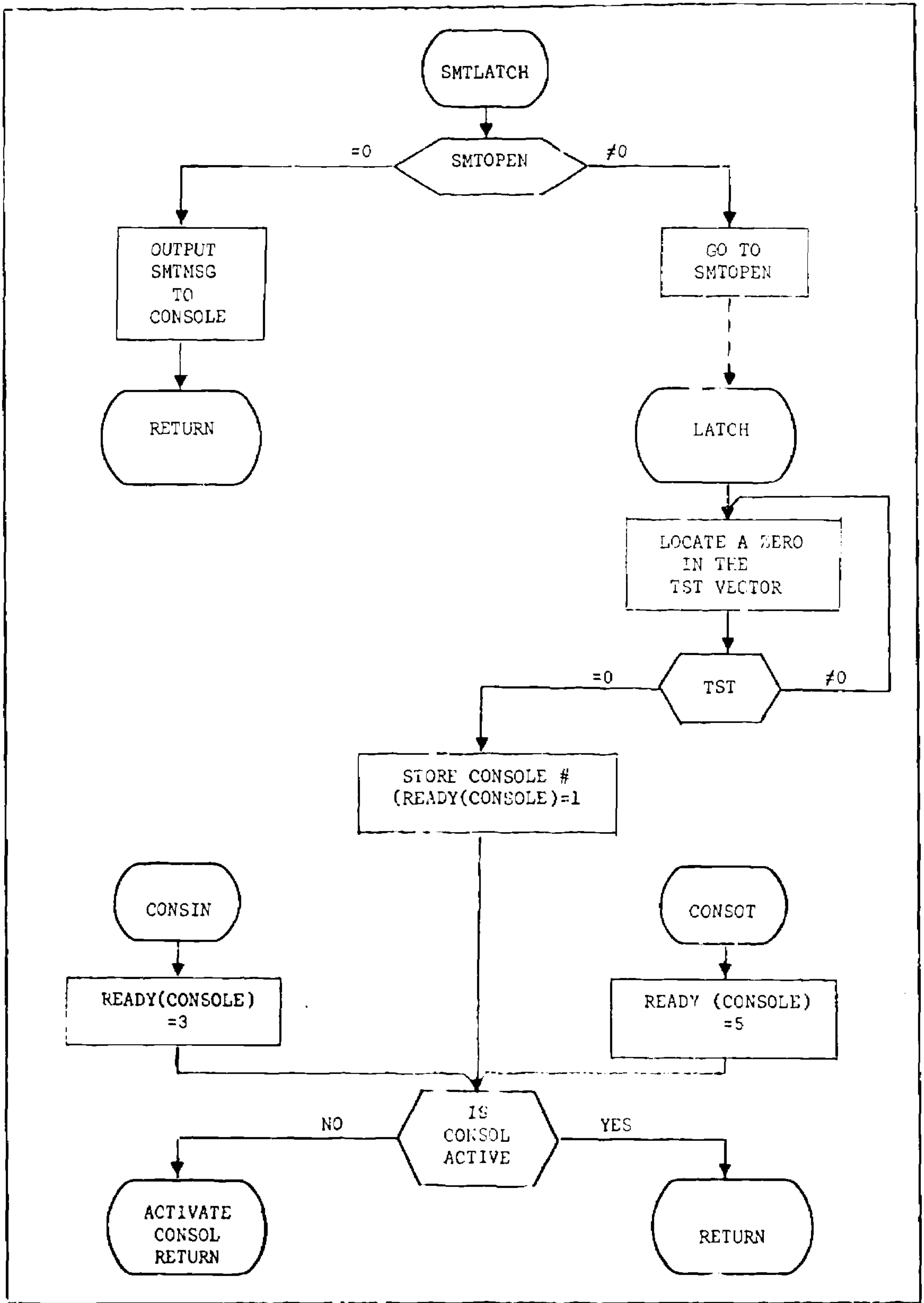
Routine CONSIN is similar to LATCH; when a console is released to a user, the supervisor needs the name of a routine to call when the transmission from the user is complete as well as a place to put the transmission. CONSIN is that routine. The supervisor tells CONSIN the name of the console which interrupted; CONSIN then changes the READY flag for the console (from 2) to 3 and insures that CONSOL is running.

To minimize over-all response times only one line will be set up for transmission to a console if another console also needs service. If a console needs several lines, but only one is transmitted, CONSOL will have to prepare other lines at a later time. To do this on an interrupt basis, routine CONSOT is called by the supervisor after transmission of a line to a console if that console requires more information.

All of these routines consist of fewer than a hundred instructions and take less than a millisecond to execute. Fast response to the changes made in the READY table is insured, since CONSOL tests the flags after each line of a transmission is complete. The test for a console needing attention is less than fifteen microseconds if no console needs attention (assuming ten on-line consoles). Since the test is so fast, frequent repetition is not expensive.

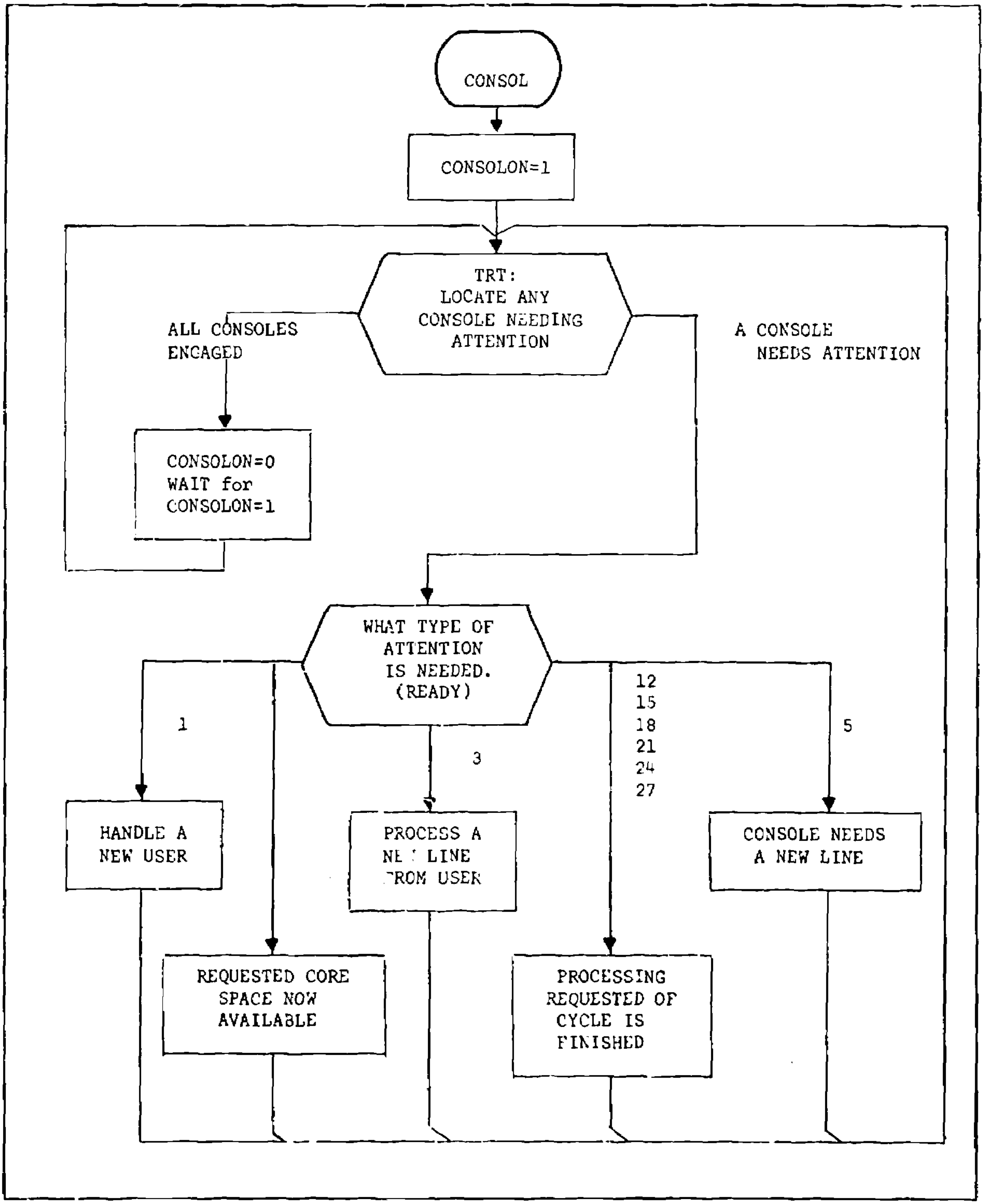
E) CONSOL as a Traffic Controller

In basic terms, CONSOL uses the TRT instruction to select a console which has a need and then satisfies the needs of that console at least temporarily. CONSOL then uses the TRT again to select another console. Eventually all console needs will be satisfied and CONSOL will retire to permit other processes to use the CPU; one of these processes will most likely be CYCLE. When CYCLE has completed a request for a user, or a set of requests, CYCLE will ask the supervisor to restart CONSOL, and, by so doing, suspend itself in real-time (but not in process-time). Alternatively, the completion of a user line at a console will result in an interrupt-initiated call to CONSIN or LATCH which can wake-up CONSOL. Effectively then, CONSOL uses the TRT instruction to facilitate a traffic direction problem.



SMTLATCH, LATCH, CONSIN, and CONSOT

Fig. 7


CONSOL
CONSOLON=1
TRT:
LOCATE ANY
CONSOLE NEEDING
ATTENTION
ALL CONSOLES
ENGAGED
A CONSOLE
NEEDS ATTENTION
CONSOLON=0
WAIT for
CONSOLON=1
WHAT TYPE OF
ATTENTION
IS NEEDED.
(READY)
1
3
12
15
18
21
24
27
5
HANDLE A
NEW USER
PROCESS A
NE : LINE
FROM USER
CONSOLE NEEDS
A NEW LINE
REQUESTED CORE
SPACE NOW
AVAILABLE
PROCESSING
REQUESTED OF
CYCLE IS
FINISHED

CONSCL

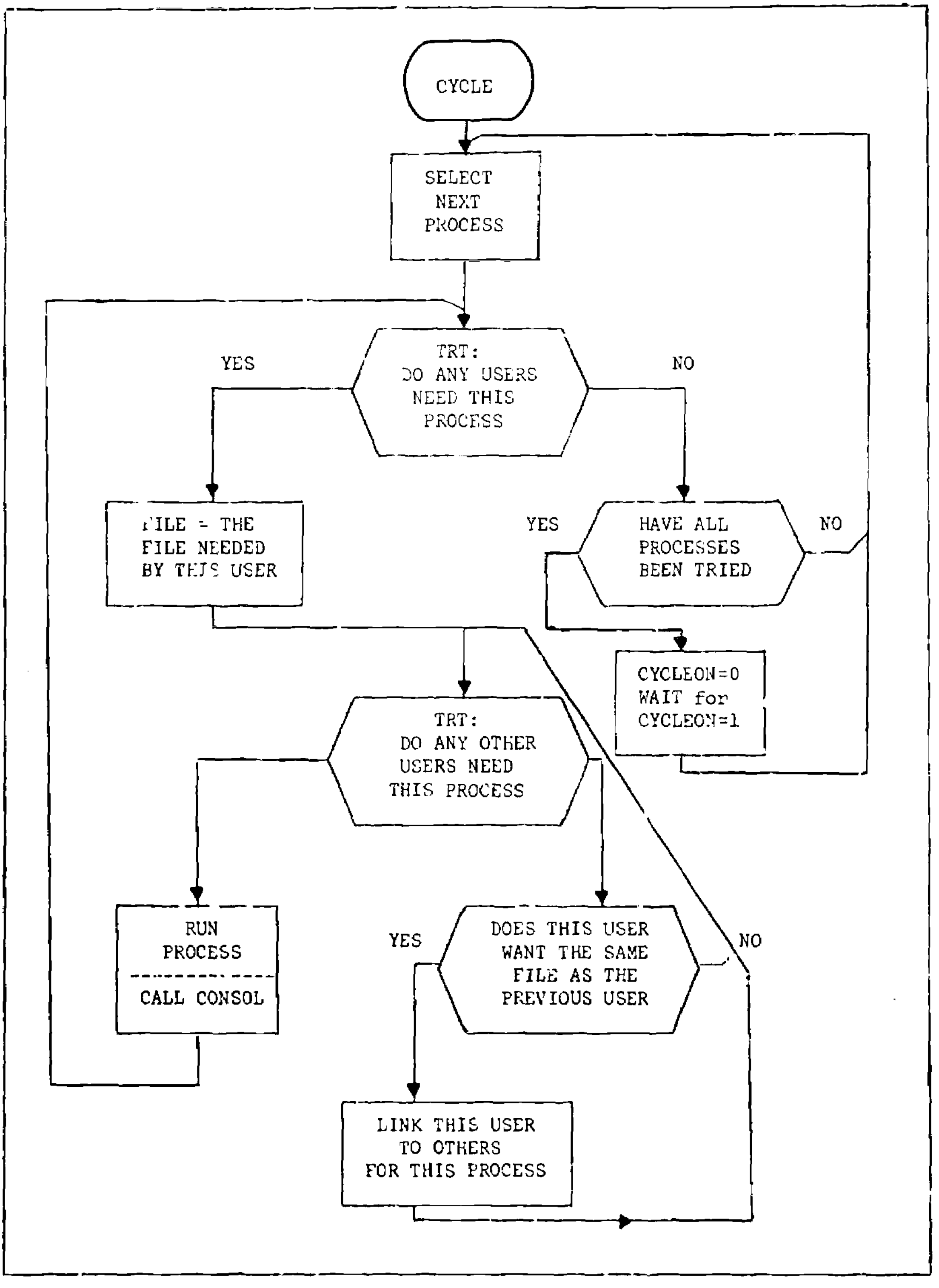
Fig. 8

It is apparent from a scan of various possible needs that some are more urgent than others. For CONSOL, however, needs are satisfied so quickly that the arbitrary selection of the console highest in the SOCCB is adequate. CONSOL works so fast that even if the 256 users were on-line and all had a need at the same instant, and the first user were serviced first, the last user would be satisfied before the transmission to the first user was complete. In actuality, in most cases, only one user will need service at any given time. The obvious exception to this is after CYCLE completes a task — at that time, several consoles will need transmissions. It is immaterial, however, which console is satisfied first, since all consoles will be satisfied by CONSOL in much less time that was taken by CYCLE.

F) A Detailed View of CYCLE

In contrast to CONSOL, CYCLE follows a strict pattern in deciding what to do. Like CONSOL, CYCLE uses the TRT instruction but CYCLE decides what process to do first. Then it sees which consoles need that process. If no console needs that process, CYCLE tries the next process in its list of processes. To permit on-line access to more than one collection for test purposes, or access by sophisticated users with special needs, each process is run for all consoles that request one collection and then for all consoles that require another collection. This is illustrated in Fig. 9.

Some object processes started by CYCLE are standard programs used for batch experimentation; text cracking, centroid tree searching, document correlation, and query redefinition. The processes unique to the on-line system divide into two classes — those that access files for the user and those that service CONSOL. There are presently two programs of


CYCLE
SELECT NEXT PROCESS
TRT: DO ANY USERS NEED THIS PROCESS
YES
NO
FILE = THE FILE NEEDED BY THIS USER
YES
HAVE ALL PROCESSES BEEN TRIED
NO
CYCLEON=0 WAIT for CYCLEON=1
TRT: DO ANY OTHER USERS NEED THIS PROCESS
RUN PROCESS
CALL CONSOL
YES
DOES THIS USER WANT THE SAME FILE AS THE PREVIOUS USER
NO
LINK THIS USER TO OTHERS FOR THIS PROCESS

CYCLE

Fig. 9

the first type: to display pre-search information, e.g. thesaurus categories, and to display post-search material, e.g. abstracts. Since CONSOL operates on an interrupt basis, it cannot allocate resources for itself. However, CONSOL does need to be able to obtain core storage space on demand. To provide this, CYCLE can be asked to allocate storage for a console and return control to CYCLE.

From the flowchart for CYCLE shown in Fig. 9, it can be seen that CYCLE restarts CONSOL without testing if CONSOL is running. This is possible since CYCLE can use the CPU only when CONSOL is inactive.

6. Summary

On-line information retrieval is implemented by two co-routines, CONSOL and CYCLE. The former operates in the real-time of the console user providing rapid response; the latter in the process-time inherent in any routine which needs to access auxiliary storage providing realistic costs for work done. The two routines communicate through a single area of mutually known core.

This system should prove adequate for both experimentation and real-time use in a library, for both the novice user and the sophisticated researcher with the complex problem.

Appendix

TRT READY, TABLEn (Instruction image, n=1, 2, 3 or 4)	
Location:	READY + 0 1 2 3 4 5 6 7 8
Contents:	5 4 3 4 2 0 0 0 1
Location:	TABLE1 + 0 1 2 3 4 5 6
Contents:	0 0 0 6 0 0 0
Location:	TABLE2 + 0 1 2 3 4 5 6
Contents:	0 0 0 0 9 0 0
Location:	TABLE3 + 0 1 2 3 4 5 6
Contents:	0 2 3 0 0 0 0
Location:	TABLE4 + 0 1 2 3 4 5 6
Contents:	0 8 0 0 0 0 0

Execution:	1st			2nd			3rd
Register:	0	1	cc	0	1	cc	cc
n							
1	6	(READY)+2	1			0	
2	9	(READY)+1	1	9	(READY)+3	1	0
3	3	(READY)+4	1	2	(READY)+6	1	0
4	8	(READY)+6	2				

(READY) means the address of READY ; cc = condition code

The Effects of the Translate and Test Instruction (TRT) When the Vector READY is Entered Against Several Tables

XV. Template Analysis in a Conversational System

S. F. Weiss

Abstract

This study presents a discussion of natural language conversational systems. The use of natural language rather than fixed format input in such a system makes possible the implementation of a natural dialogue system, and renders the system available to a wide range of users. A set of goals for such a system is presented. These include the provision of fast responses, usable by all levels of users, and the use of intellectual aids such as tutorials.

An experimental conversational system which meets these goals is implemented using a template analysis process. Template analysis is used not only to analyze natural language input, but also to control the overall operation of the process. Experiments with a number of users show that the system is easy to utilize and provides accurate analyses. A detailed discussion of both user and system performance is presented.

1. Motivation

Programs and data are normally entered into a computer in a batch processing mode. However, the recent trend in computer system design has been toward the development of large time shared systems which give a number of users simultaneous on-line access to the computer. This makes possible the implementation of conversational programs which permit real-time man-machine dialogues. Such conversational programs are both useful and necessary to cope with the ever expanding complexity of computerized data processing tasks. Consider for example, an on-line programming language such as APL. The ability to test amd debug a program on-line is an aid to the programmer. Errors are more easily located and may be corrected immediately. In addition, on-line data entry allows the programmer to adjust parameters and data while the program is running in order to get the desired results.

Conversational programs are also useful in all forms of language processing and expecially in information retrieval. Consider for example a case in which a natural language analysis program encounters an unresolvable ambiguity. In the batch mode, the program would be forced either to give up or to proceed using the multiple interpretations. But in a conversational mode, the system can ask the user for clarification and then proceed with perfect information as is shown in the example in Fig. 1.

U: TYPE 2 GRAMMARS

S: YOU HAVE USED TYPE AMBIGUOUSLY. PLEASE SPECIFY:

A. PRINTING

B. VARIETY

U: B

S: PROCEED

User Disambiguation

Fig. 1

In information retrieval the applicability of conversational programs is very broad. It is the only way to make the retrieval operation fast enough for practical use. In addition it permits the user to see results immediately and adjust his query and other search parameters to tailor the performance to his exact needs. The conversational mode is also the best framework in which to implement the relevance feedback process [11,24]. In general the conversational facility is an extremely powerful information retrieval tool.

Section 2 of this study discusses some existing on-line systems. Most of them require a fixed format input. But the current trend in information processing is toward natural language input. Not only does this permit the treatment of documents and queries in their original form, but it also makes the on-line facility available to a broad spectrum of potential users. This is especially important since on-line systems permit remote access from places such as libraries and schools which are not inhabited strictly by computer people. This study discusses conversational systems in general and presents a natural language facility for information retrieval.

There are four basic goals which any such natural language conversational system should meet. First, the system obviously must accept natural language input. Second, it must provide fast response. Users tend to become impatient if the delay between the submission of a command and the system's response exceeds more than a few seconds. Third, the system should be usable by all levels of users. Inexperienced users should be able to perform useful work. At the same time the system must not hamper the expert with excessive verbosity and unwanted material. And finally, the system should provide some intellectual aids such as tutorials and prompts which can help the user conduct a useful dialogue.

2. Some Existing Conversational Systems

Many conversational systems are currently in operation. Most are part of a larger implementation such as an information retrieval system. But a few such as ELIZA are designed solely to perform conversation. The major differences among the conversational aspects of the various systems is in the amount of man-machine interaction permitted. In some systems the on-line input is not far removed from batch input and the user has little control over the running of the process. At the other extreme are systems in which the user is directly linked to the process and is continuously in command of program operation. The discussion of on-line systems presented below is roughly in order of increasing complexity of dialogue.

The most basic type of conversation consists of a simple user input which results in some appropriate system action being performed. RECON [16], DIALOG II [29], TIP [15], and AUTONOTE [22] are representative of this type of conversation. In RECON for example, the user presses a button which indicates the desired operation and then types the operands on the console. In the other systems the user types the operator name followed by operands. Thus all these processes require a fixed input format. In addition, should the user become lost or confused, the systems cannot supply any intellectual aid to help him out. One type of user aid, the tutorial, is a feature of the AUDACIOUS system [2]. In addition to the normal operator-operand commands like those above, AUDACIOUS permits two special commands: HELP and PUNT. In response to these, the system produces a tutorial message appropriate to the user's position in the dialogue. In this way the confused user can receive help.

A second type of intellectual aid is the prompt. SPIRES [21] is an example of a system which uses the prompting feature. Unlike tutorials, prompts are presented without user request. Their purpose is to indicate to the user what type of information is to be specified in the current input. However, since prompts are presented without a user request, they can sometimes be a nuisance to the expert user. All the conversational systems presented thus far share two attributes. First, they all require fixed form input. And second, they are all information retrieval systems and hence the conversational operation was not the prime consideration in their development. The systems discussed in the next few paragraphs are designed basically to conduct conversation in natural language.

Probably the most famous natural language conversational system in Weizenbaum's ELIZA [34]. The program conducts a coherent dialogue with the user much like that between a psychotherapist and his patient. Inputs are searched for the presence of certain keywords and structures. These indicate the type of output appropriate to the input. For each input form there is more than one allowable response. ELIZA cycles through this set thus eliminating repetition and producing a more realistic looking conversation. The approach to conversation used in the system presented later in this study is similar to the ELIZA concept.

Another area of usefulness for conversational capabilities is in computer assisted instruction. One such conversational CAI system is Bolt's Socratic Instruction [6]. Its operation is basically an extension of the techniques outlined for ELIZA. Like ELIZA, the Socratic Instructor uses the user position in the dialogue along with the input to determine the proper response. In addition, the Socratic Instructor remembers all previous user inputs and dialogue points. These are also used in output determination.

Most conversational systems in existence today are implemented by basically ad hoc programming methods. This is not unusual for a fairly new area such as conversational programs. However, as on-line systems become more common, higher level implementation processes must be developed. One such process already in existence is the LYRIC system developed by Silvern [26]. This is a programming language for describing conversational CAI programs. With processes such as this, the conversational implementer is relieved of some of the ugly programming details in much the same way as a compiler-compiler aids the systems programmer.

The conversational systems presented here by no means constitute the complete set. They are, however, representative of most systems. It appears that systems such as TIP and SPIRES which perform efficient on-line information retrieval require highly structured input format. On the other hand those such as ELIZA which permit natural language input have a very weak concept of understanding. It would be desirable to develop a system which combines the best attributes of both; that is, a fast and accurate information system which allows natural language input. This is the topic of the following sections.

3. Goals for a Proposed Conversational System

This section discusses the design considerations that go into the development of a new conversational information retrieval system. Some elements of the new system are drawn from existing facilities while others are new. The primary goal of this system is to allow a user to conduct a natural language dialogue with the system. The only limitation is that the input be restricted to an information retrieval context. Not only should the user be allowed to specify natural language commands, but also there should be no restriction on the number of commands per line as there are in most other conversational systems. An input such as

> USE THE COSINE CORRELATION ON THE CRANFIELD COLLECTION.

should be perfectly legal. Of course there may be some inputs for which natural language is impossible or impractical and a fixed format input must be used. For example, the user should [illegible] required to specify a fixed form "SIGNOFF" in order to prevent accidental termination of the conversation. But these formatted inputs should be kept to a minimum. Another goal for this system is to be able to resolve automatically ambiguities occurring in the user's input. In addition the system must meet the requirements specified in section 1. These include providing fast response, being usable by all levels of users, and providing intellectual aids such as tutorials and prompting.

This proposal makes demands on the user as well as the system. First, the basis for learning the system is a manual. It would be aesthetically pleasing to allow the system itself to contain a computer aided instruction facility (CAI) which would make the system completely self-contained. Unfortunately this is impractical. Successful CAI requires concentrated and frequent exposure to the teaching medium. It appears that the typical information retrieval user dialogue will be both brief and fairly infrequent. Also, trying to teach the user at the console unnecessarily ties up the facilities. Thus an off-line approach to learning the system seems more reasonable. While no CAI facility is provided, the system should offer a prompting option by which a user can be led step by step, through a simple retrieval process. In this way the user may learn something about the system while actually performing useful retrieval work. The user's manual for this system is divided into several sections. Each deals with system use in progressively greater detail. A user need only read those parts which satisfy his particular need. A casual user who wants only simple retrieval operations using system defaults, has to read only a few pages. And the prompting facility can be used with only a paragraph or so of instruction.

Another user problem that must be treated is the separation of novices and experts. As is often the case, conversational systems are handled by users with widely varying degrees of expertness. The system should neither hamper the expert with excessive verbosity nor hinder the novice with obscure and terse responses. Some systems compromise and use a "middle of the road" approach, but this satisfies no one. Other systems have multiple sets of dialogue scripts. A user is classified as having a particular level of proficiency and he receives the dialogue appropriate to that level. But this too can lead to problems. In any large facility such as an information retrieval system, it is entirely possible for a user to be very proficient in some but not all areas of the system. Classifying him strictly as a novice or expert is wrong in both cases. To solve this problem, the proposed system uses an implicit rather than explicit separation of novice and expert. This is accomplished by allowing access to options only when the user asks for them. Thus the more the user knows about the system, the more facilities he has at his disposal. The novice is thereby protected from options which he does not understand. Tutorials are also presented only on request. Because of this only a single set of tutorials is needed and they can be reasonably long and clear. The expert user who does not ask for a tutorial need never see any and thus is not hindered by them. The only manifestation of the novice facilities that an expert must see is the short question:

Do you need help in using the system?

This appears immediately after signon. Even this can be eliminated by allowing a user status file to be stored between system uses. Upon signing on, the user's status file is read and appropriate parameters, including his negative answer to the above question, are set.

A few other characteristics of the proposed system also help in the proper handling of both novice and expert users. These are the multi-step processing technique and the ability to compound commands on a single line. An expert, for example, can put several system commands into a single input thus saving time and effort. The same commands may also be split on a number of lines for greater clarity. This and the multi-step process are discussed in greater detail in section 4.

One final goal of the proposed system is [illegible]entation of useful tutorials. These messages must be easily available so that even the most confused user can get help. One simple method is to use a single question mark "?" as the tutorial request. The tutorials must reflect the specific place in the dialogue where they are called. In addition, they must take into consideration the commands and options that the user has already specified. Tutorials are also useful in treating errors. When an erroneous input is detected, the system automatically produces a tutorial appropriate to the place where the error occurs. The incorrect input is an implicit indication that the user needs help and thus the tutorial is appropriate at that point.

The design considerations presented in this section are basically nontechnical. They stem from an effort to satisfy within practical limits the basic conversational needs of the largest possible user population. The next section presents a discussion of the actual implementation of such a system.

4. Implementation of the Conversational System

This section discusses the implementation of the conversational system. The major obstacle in the process is the fact that the Cornell University Computing Center has at present, no facilities for user implementation of on-line systems. The programs thus must all be run in the conventional manner with batched input. This poses no real problem in the design and operation of the system except in the area of testing it on real users. But even this can be circumvented with adequate simulation.

A) Capabilities

The conversational system is designed to perform SMART-like information retrieval operations. The capabilities built into the present system include specification of a correlation coefficient, search strategy and collection to be used. The first two of these are provided with default values that are used if nothing is explicitly specified by the user. There is provision for submitting a query containing a number of data base entry point references (subject, date, journal, and author). A search can be initiated and the user can request to see any number of retrieved documents. In addition to these information retrieval operations, the user has available some commands to the conversational system itself. These include requesting a tutorial, asking to be guided through a retrieval operation, and signing on or off. A few other information retrieval operations, most notably relevance feedback, are deliberately omitted, since the system is designed to test the conversational and natural language capabilities, and not to retest the information retrieval techniques. The set of capabilities is selected as typical of the inputs, outputs and internal processes required in a larger system. Also relevance feedback is not conductive to handling in natural language. While a user might introduce a natural language input which indicates his desire to perform relevance feedback, the actual submission of relevancy judgements is best handled in a fixed format. Relevance feedback and a few other capabilities would add little to the significance of system experimentation and hence are omitted.

B) Input Conventions

While it is the aim of this system to allow natural language input, there are a few places where the use of natural language is impractical. This is usually caused by the physical characteristics of the conversational system or information retrieval in general. One such instance is in setting off a query from other types of input. A query may deal with any subject area. For example it could ask for information about some aspect of a conversational system. It could thus be indistinguishable from a legal system command. For this reason, the user rather than the system, must perform the discrimination between queries and commands. This is accomplished by simply prefacing each query with "QUERY" or "Q". This adds little to user effort and eliminates what might be an impossible system task. Another area where fixed format is necessary is in search initiation. Unlike other operations in a conversational system which require only a few computer cycles, the search is relatively costly in computer time. It is therefore desirable to avoid uncalled for searches. Also, searches should not be initiated until the user is satisfied with his query and search specifications. For these reasons, searches are performed only upon an explicit signal ("GOSEARCH") from the user. A third fixed format input is the request for a tutorial. This is accomplished by typing a single question mark ("?"). This is done strictly for user convenience. In this way, even the most confused user can receive a message appropriate to his present dialogue position. Tutorials are also automatically generated when a user introduces an incorrect input. The final fixed form input is the SIGNON command. In an actual on-line implementation, it is quite possible that this command will be handled by a supervisor program which controls all on-line operations. Thus the natural language analysis facility may not be present to process this input. The remainder of the inputs may be posed in natural English.

C) The Structure of the Process

The structure of the conversational system may be viewed as a graph. The nodes represent user decision points and the edges represent possible alternatives and system actions. As the user progresses through his dialogue, he moves from node to node in the graph. The action is much like that of a finite automaton. At every point in the dialogue, the user is at some system node. The combination of this current node and the user's input at that point determine the action to be performed (analogous to the output of the automaton) and the node to which control is passed after the action is completed. This strategy allows the system to be throught of as a set of modular units. Each unit corresponds to a node and each has associated with it the subset of inputs that are legal at that point, as well as the associated actions. The input processing is thus greatly simplified since at each node the system need only test for those inputs that are legal. All other inputs are illegal even though they might be acceptable at some other point in the dialogue. The modular approach also facilitates some degree of disambiguation. Some inputs are ambiguous when considered with respect to the total set of system inputs. However, many become unambiguous within the context of a single node. The simplest example is the tutorial request ("?"). The question mark by itself is not enough to determine which of the many tutorials is desired. But the combination of the question mark and the current node performs the disambiguation and the proper message is presented.

D) Template Analysis in the Conversational System

There are two main jobs to be performed in a natural language conversational system. The first is the natural language analysis required to transform the input to a machine-usable form. The second job is bookkeeping. The system must keep track of the user's present position in the dialogue, the legal inputs as well as the successor node associated with each input. It seems relatively clear that the template analysis process introduced by Weiss [31] is sufficient to handle the natural language analysis task. The expected input consists of queries and system commands coming from some sort of on-line terminal. They thus conform exactly to the user restricted input for which template analysis is designed. While more complex systems would produce a more rigorous analysis of the input, template analysis can provide all the information that is needed from the input and at a considerable saving in time over other methods. Thus template analysis appears to be the ideal natural language anlaysis technique for this application.

Upon first analysis the bookkeeping task seems outside the realm of template analysis. But actually, the most efficient way in which to implement this task is to imbed it within the template analysis structure. This is done as follows. Each template is applicable to only one node, which is called its host node. This is indicated by appending the host node number to the template concept numbers. Since template concept numbers range from 11 to 999, this appending can be accomplished by adding the desired node number times 1000 to the concept number. Each template contains a set of concept numbers, a key word, and a link to an action routine that is to be executed if that template is matched. Some additional information must be added for the conversational application. Each template must contain a next node: immediate (NNI) number which tells the node to which control is to be transferred immediately after execution of the associated template action. It is sometimes useful to defer transferring to a new node until all possible executions of the template action have been performed. For example in cases where a number of similar pieces of information must be picked up from one input. In this case, NNI refers to the host node. A second value, the next node: final (NNF) then indicates the node to which control is transferred after all actions at the current node are complete. In the examples in Fig. 2 below, template A and B are both applicable only in node 5, and both match the same input substring. After matching, however, template A calls action routine 51, and control is then immediately transferred to node 2. Template B causes action 55 to be performed and control remains at node 5. Finally, after all possible node 5 matches have been processed, control passes to node 3. In cases such as A where NNI causes a transfer to a node other than its own, the NNF value is ignored.

	NNI	NNF	ACTION	TEMPLATE CONCEPTS
A	2	-	51	5011, 5012, 5013
B	5	3	55	5011, 5012, 5013

Sample Conversational Templates

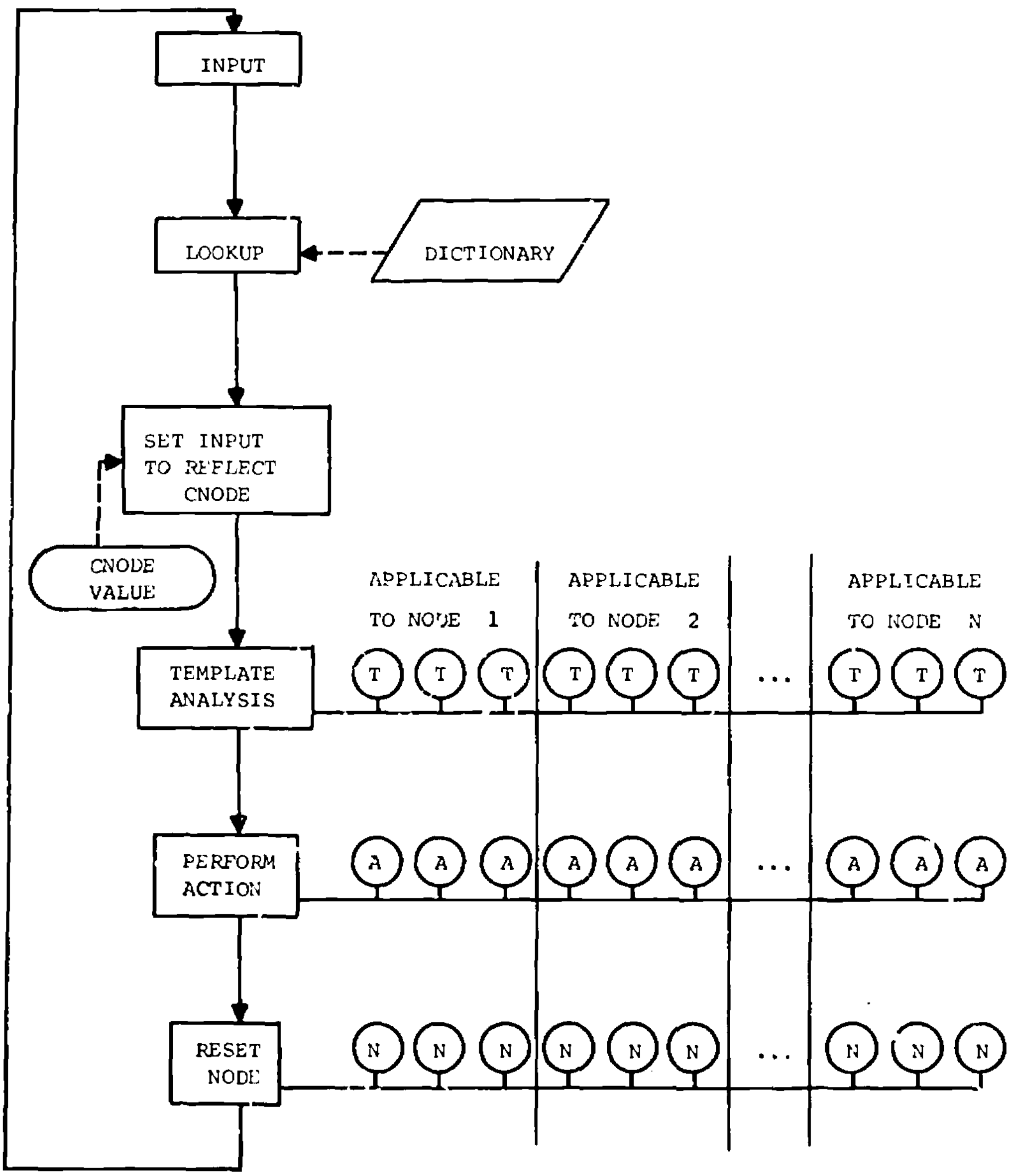
Fig. 2

In order to match the proper templates, the input must be made to reflect the current node (CNODE) in the dialogue. Upon reading an input, the current node times 1000 is added onto each input concept. Also, after every node change, the old node number is stripped off the input and the new node times 1000 added on. Thus the input reflects the current node in exactly the same way in which the templates reflect their host nodes and hence proper matching occurs. In this way the template process itself keeps track of the current node, the legal inputs for each node and the successor node function. This operation is summarized in the schematic in Fig. 3. An input is read and each word is assigned a numeric concept by a dictionary lookup. The input is then set to reflect the current node i . A scan is made of the entire template set in search of a match. However, only those templates whose host node is i have any chance of matching. If a match in this subset is found, the associated action is performed and the next node path is followed.

Fig. 4 indicates the node structure of the conversational system. Node 2 is the supervisor. After the initial signon phase, operations generally start and end in node 2. Most operations are two step processes. First, in node 2, the input is analyzed and the type of operation that it specifies is determined. Control then passes to the appropriate new node. Second, in this new node, the exact operation is determined and executed. Control is then returned to node 2. As an example consider the input

USE THE COSINE CORRELATION.

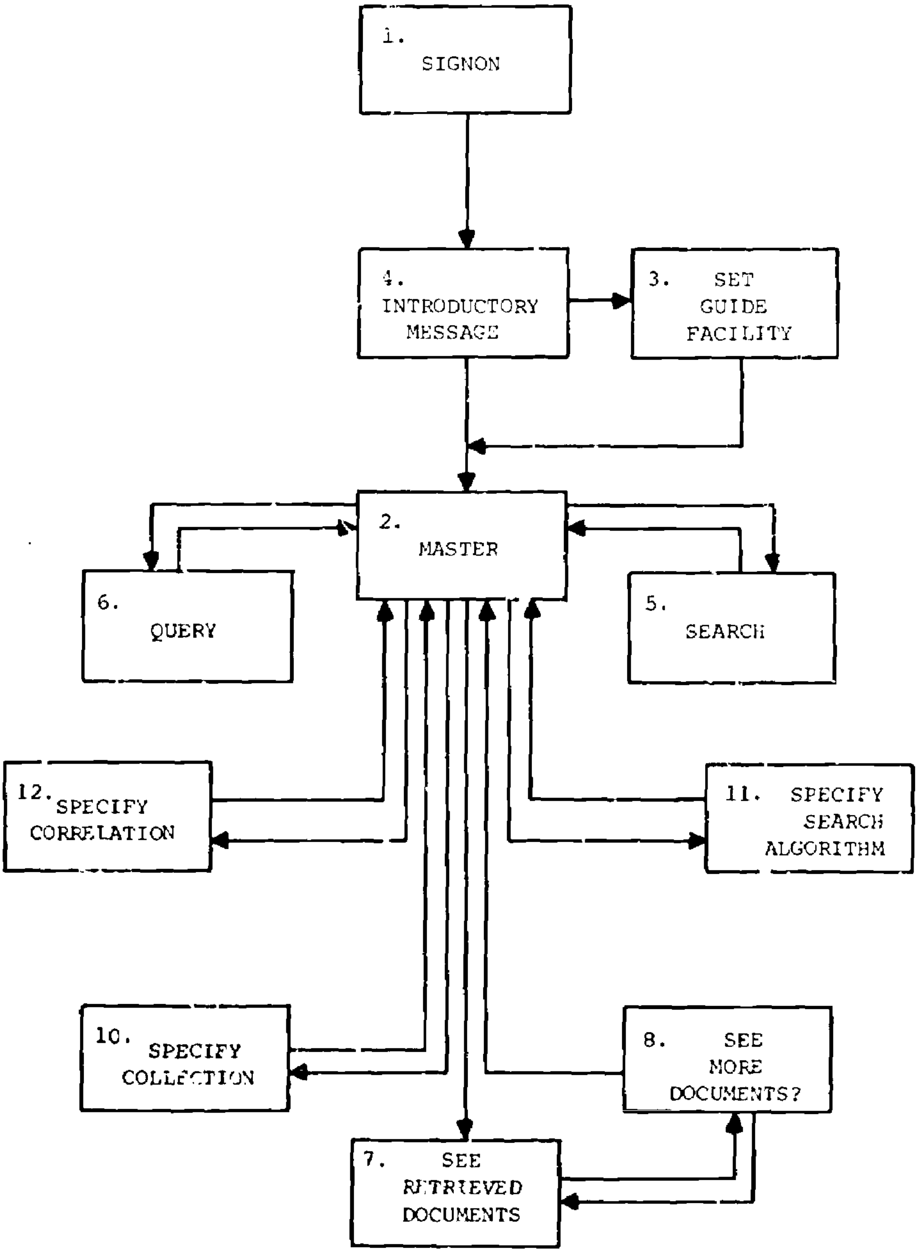
In node 2, it is determined that a correlation is to be specified and control passes to node 12. In node 12, the specific



T=TEMPLATE
A=ACTION
N=NEXT NODE

Schematic of Conversational Operation

Figure 3



Conversational Node Structure

Figure 4

correlation coefficient (i.e. cosine) is detected and noted. Control then goes back to node 2.

There is no necessity that the commands for two step operations appear on the same input line. For example, simply typing "CORRELATION" causes a transfer of control from node 2 to 12. The system then waits in 12 for further instructions. Strictly for the sake of convenience a special feature is used in cases like this. Whenever the system finds itself waiting in a node other than 2 it knows that an incomplete input has been entered. A special routine is therefore called to print a message appropriate to the current node. This aids the user in completing the input as is shown below. In this example and in all other samples of conversational scripts, user input is identified by a leading "U:".

U: CORRELATION

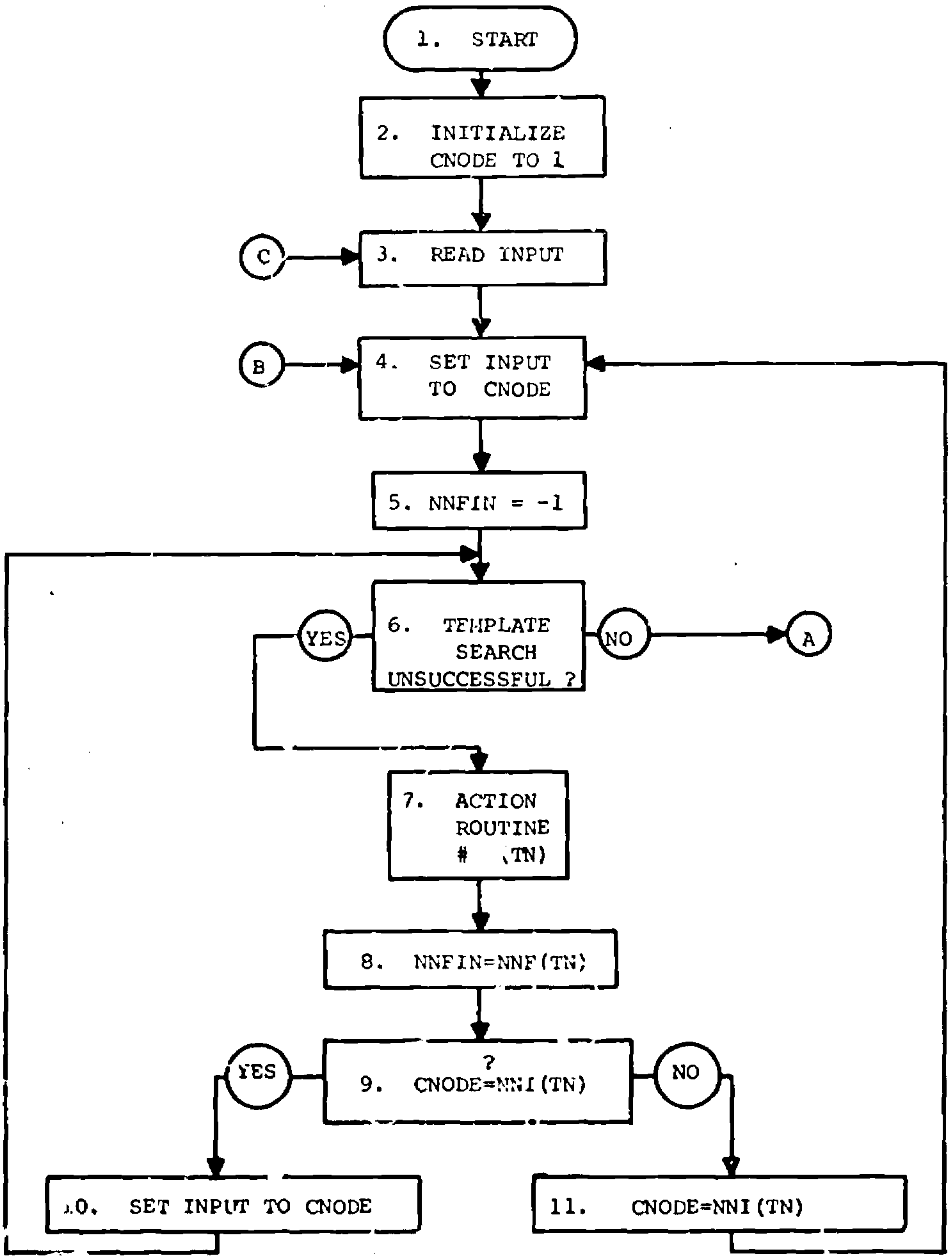
SPECIFY A CORRELATION

U: COSINE

Not only can inputs be spread out over several lines, several inputs can also be compounded onto a single line. For example

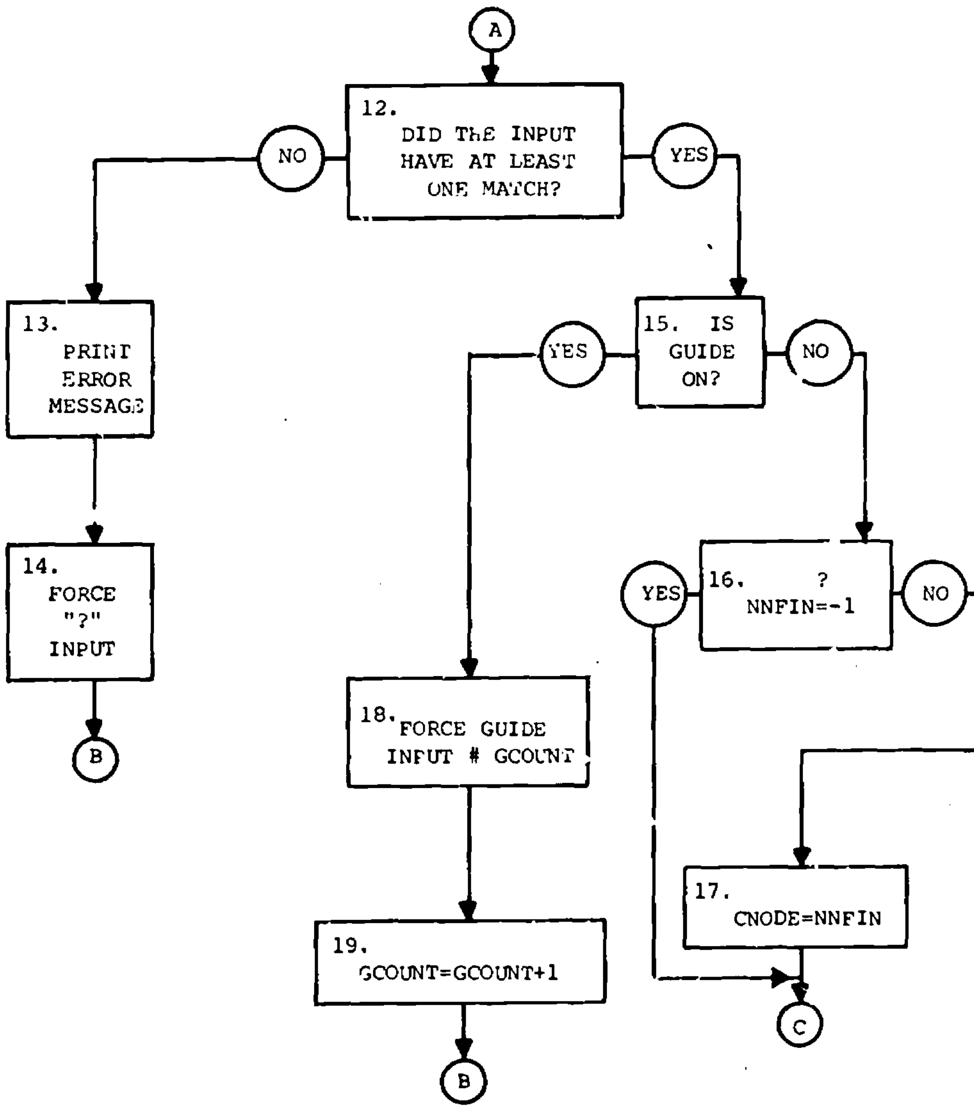
U: PERFORM A FULL SEARCH ON THE PHYSICS COLLECTION WITH THE COSINE CORRELATION.

As is seen in the detailed flow chart in Fig. 5, once an input is read, it is processed repeatedly until all valid template matches are exhausted. This results in an exit from box 6 via failure. Since this same exit is taken regardless of how many



Conversational Control Algorithm

Figure 5



NOTE: NNFIN is the next node: final value. It is initialized to -1 before template matching begins. If no template matches are found, it will still be -1 at box 16. This indicates that control is to remain at the current node.

Conversational Control Algorithm

Figure 5 (Cond.)

or few, template matches occur in the input, a test must be made to see if at least one match occurs (box 12). If not, the input is not valid and a diagnostic must be presented to the user. The system prints a short general error message, erases the current input and replaces it by a question mark. Control is then passed back to the input analysis section. This results in the appropriate tutorial being shown to the user. This process of supplying diagnostics by allowing the system to force in a special input and then treating this as a normal user input is also used in the implementation of the guide facility which is discussed below.

E) The Guide Facility

In the original proposal for this system, a desire is expressed to provide a prompting facility to guide a novice user, step by step, through an actual retrieval operation. When a user signs onto this conversation system, he receives a brief introductory message:

Do you need help in using this system?

If the user is familiar with the system he can simply answer NO and he sees no more of the prompting script. If his answer is YES, he receives a somewhat longer introduction to the system (See Fig. 6) and is then asked if he wishes to be guided through a retrieval operation. If not, the system operates normally and no prompting is given. If on the other hand, his answer to the second question is YES, the guide facility is turned on. The guide subroutine has a set of special strings the form:

<general operation> ?

These include for example:

CORRELATION ?

SEARCH ?

QUERY ?

etc.

Each time the guide subroutine is called (see Fig. 5, boxes 18 and 19) i orces its ith string into the input area, increases i by one, and transfers control back to the input analyzer. These special inputs have the effect of performing the first half of a two step operation and then generating a tutorial. All the user has to do is respond in turn to each tutorial thus completing the second half of the two step process. When the guided retrieval process is finished, i is reset to one and the user is asked if he wants to be guided again.

F) Tutorials

There is a tutorial associated with each system node. When the use. types a question mark, he is given the tutorial appropriate to his current node. The tutorials for all nodes except 2 provide instruction on the specific type of input expected. Unlike other nodes which have a very limited legal input set, almost all options are available from node 2. A different and more detailed form of tutorial message is necessitated in this case. The node 2 tutorial consists of two arts: the present status and the available options. The status report provides a summary of the specifications that the user has already made. The available options are presented as a list of tasks that are currently legal. Each option in the list has an identifying letter so that the user may pick it simply by typing the letter.

Fig. 6 shows some actual scripts produced by the conversational system with various levels of users ranging from novices who use the guide facility (Fig. 6A) to highly knowledgeable experts. The scripts include both correct and incorrect inputs as well as the various tutorial forms. The program which performs the conversation is written in G-level FORTRAN-IV. It consists of about 1300 FORTRAN statements and includes 35 subroutines and entry points. The program uses two output streams. One is used for diagnostic output. This is useful in debugging and in determining if the program's internal operations are working properly. The second stream is the conversational output which would be displayed on the user console. Timing statistics for the system are presented in section 5.

5. Experimentation

The experimentation tests the system with actual users and analyzes the results both with respect to system performance and user performance. The user population used represents a fairly broad spectrum of potential conversational system users with varying degrees of knowledge of information retrieval. The experiment for each user consists of two parts. First, the user is given a brief introduction to the system. Basically he is told that this is a conversational information retrieval system

```
U:  SIGNON
SMART ON-LINE...DO YOU NEED HELP IN USING THE SYSTEM?
U:  YES
THIS IS A NATURAL LANGUAGE CONVERSATIONAL
INFORMATION RETRIEVAL SYSTEM.  THE SYSTEM RETRIEVES
DOCUMENTS IN RESPONSE TO ENGLISH QUESTIONS.  YOU MUST SELECT
A DATA COLLECTION TO BE SEARCHED.  IN ADDITION YOU CAN
SELECT SEARCH STRATEGIES AND CORRELATION OR ALLOW SYSTEM
DEFAULTS.  IF YOU NEED HELP AT ANY TIME IN YOUR DIALOGUE,
TYPE "?" AND YOU WILL RECEIVE A TUTORIAL APPROPRIATE TO YOUR
PRESENT POSITION IN THE DIALOGUE.  FOR FULL INFORMATION
CONSULT THE MANUAL.

DO YOU WISH TO BE GUIDED THROUGH A RETRIEVAL PROCESS?
U:  YES
THE FOLLOWING COLLECTIONS ARE AVAILABLE FOR SEARCH
(SELECT BY NAME OR LETTER).
     A.  ENGINEERING
     B.  BIOLOGY
     C.  LITERATURE
     D.  CHEMISTRY
     E.  PHYSICS
U:  A
THE FOLLOWING CORRELATIONS ARE AVAILABLE.  CONSULT
MANUAL FOR FORMULAS.
     A.  COSINE (DEFAULT)
     B.  OVERLAP
U:  COSINE
THE FOLLOWING SEARCH STRATEGIES ARE AVAILABLE.
     A.  FULL SEARCH (CONSULT MANUAL BEFORE USING)
     B.  CLUSTERED (DEFAULT)
U:  CLUSTER
YOU MAY NOW ENTER YOUR QUESTION.  IT CAN BE IN NORMAL
ENGLISH AND MAY CONTAIN SPECIFICATIONS FOR DATES (YEAR),
JOURNAL, AND AUTHOR.
U:  INFORMATION ON ORDERING OF RECURSIVE FUNCTIONS BY THE
U:  DIFFICULTY OF COMPUTATION
YOU MAY NOW INITIATE A SEARCH BY TYPING GOSEARCH
U:  GOSEARCH
SEARCH UNDER WAY, EXPECT DELAY
SEARCH COMPLETE
```

CONVERSATIONAL DIALOGUE

Fig. 6A

```
INDICATE HOW MANY DOCUMENTS YOU WANT TO SEE.
IF NON TYPE "NONE".
U:  GIVE ME 5
RANK      1 DOC      91 CORR = 0.9896705
RANK      2 DOC      11 CORR = 0.9791623
RANK      3 DOC      42 CORR = 0.8563247
RANK      4 DOC      40 CORR = 0.8002142
RANK      5 DOC      45 CORR = 0.7656324
DO YOU WANT TO SEE MORE?
U:  YES 5 MORE
RANK      6 DOC      98 CORR = 0.6665342
RANK      7 DOC      25 CORR = 0.5542100
RANK      8 DOC      32 CORR = 0.4022313
RANK      9 DOC      89 CORR = 0.3987652
RANK     10 DOC      41 CORR = 0.3654201
DO YOU WANT TO SEE MORE?
U:  NO THANK YOU
SMART ON-LINE...DO YOU NEED HELP IN USING THE SYSTEM?
U:  NO
U:  ?
PRESENT STATUS:
     SEARCH ENGINEERING COLLECTION
     COSINE CORRELATION
     CLUSTERED SEARCH
     NO QUERY HAS BEEN SPECIFIED YET

AT THIS POINT YOU MAY:
     A.   SPECIFY A DOCUMENT COLLECTION
     B.   SPECIFY A SEARCH ALGORITHM
     C.   SPECIFY A CORRELATION COEFFICIENT
     D.   SUBMIT A QUERY
     F.   LOOK AT SOME RETRIEVED DOCUMENTS
     G.   START OVER
     H.   SIGNOFF
U:  H.  SIGNOFF
SIGNOFF ACCEPTED
```

CONVERSATIONAL DIALOGUE

Fig. 6A

(cont'd)

```
U:  SIGNON
SMART ON-LINE...DO YOU NEED HELP IN USING THE SYSTEM?
U:  NO
U:  YES
INPROPER INPUT
HELP FOLLOWS
PRESENT STATUS:
     SEARCH PHYSICS COLLECTION
     COSINE CORRELATION
     FULL SEARCH
     NO QUERY HAS BEEN SPECIFIED YET

AT THIS POINT YOU MAY
     A.  SPECIFY A DOCUMENT COLLECTION
     B.  SPECIFY A SEARCH ALGORITHM
     C.  SPECIFY A CORRELATION COEFFICIENT
     D.  SUBMIT A QUERY
     F.  LOOK AT SOME RETRIEVED DOCUMENTS
     G.  START OVER
     H.  SIGNOFF
U:  Q.  1970 LASER ARTICLES
QUERY SUMMARY:  DATE = 1970
U:  4
YOU CANNOT SEE DOCUMENTS UNTIL A SEARCH HAS BEEN PERFORMED
COMMAND IGNORED
U:  GOSEARCH
SEARCH UNDER WAY, EXPECT DELAY
SEARCH COMPLETE
U:  4
RANK     1 DOC     56 CORR = 0.9828174
RANK     2 DOC      5 CORR = 0.9620456
RANK     3 DOC     81 CORR = 0.8022764
RANK     4 DOC     21 CORR = 0.6553271
DO YOU WANT TO SEE MORE?
U:  NO
U:  SIGNOFF
SIGNOFF ACCEPTED
```

CONVERSATIONAL DIALOGUE

Fig. 6B

```
U:  SIGNON
SMART ON-LINE...DO YOU NEED HELP USING THE SYSTEM?
U:  NO
U:  ?
PRESENT STATUS:
     NO COLLECTION SPECIFIED YET
     COSINE CORRELATION (BY DEFAULT)
     CLUSTERED SEARCH (BY DEFAULT)
     NO QUERY HAS BEEN SPECIFIED YET

AT THIS POINT YOU MAY:
     A.  SPECIFY A DOCUMENT COLLECTION
     B.  SPECIFY A SEARCH ALGORITHM
     C.  SPECIFY A CORRELATION COEFFICIENT
     D.  SUBMIT A QUERY
     G.  START OVER
     P.  SIGNOFF

U:  CORRELATION
PLEASE SPECIFY A CORRELATION
U:  ?
THE FOLLOWING CORRELATIONS ARE AVAIABLE FOR SEARCH.
CONSULT MANUAL FOR FORMULAS.
     A.  COSINE (DEFAULT)
     B.  OVERLAP
PLEASE SPECIFY A CORRELATICN
U:  B
U:  FULL SEARCH OF THE PHYSICS COLLECTION
U:  Q:  MU MESON PARTICLES
U:  GOSEARCH
SEARCH UNDERWAY, EXPECT DELAY
SEARCH COMPLETE
U:  I WANT 5 DOCUMENTS
RANK     1 DOC     53 CORR = 0.86547531
RANK     2 DOC     41 CORR = 0.7024357
RANK     3 DOC     49 CORR = 0.6654654
RANK     4 DOC      9 CORR = 0.5557575
RANK     5 DOC     17 CORR = 0.4302142
DO YOU WANT TO SEE MORE?
U:  NO
U:  SIGNOFF
SIGNOFF ACCEPTED
```

CONVERSATIONAL DIALOGUE

Fig. 6C

and that he must type "SIGNON" to begin. From then on, the user is on his own. The intent here is to see if the uninitiated user elects the guide option and if so, is the user successfully able to complete a retrieval operation using the guide facility? In the second experimental phase, the user tries to be more of an expert. Using information he has learned during the guided operation and some additional instruction, the user performs a second retrieval operation. This second operation is done without the aid of the guide facility. The sample scripts in Figure 6 are the actual results of these experiments with a few of the users. Results must be analyzed with respect to both system and user performance. For the most part, system performance can be measured objectively while user performance is more subjective.

A) System Performance

The basic measure of system performance is simply how many inputs are handled correctly out of the total number seen. This can be divided up since inputs arrive from several sources. Most inputs come directly from the user, but some are forced into the input area by the system itself. An input may be legal or illegal. Most illegal inputs are requests for options not accessible at the current node. If it is legal, a correct analysis is produced if the system performs the action intended by the user. For an illegal input, a correct analysis takes the form of noting the error and printing an appropriate message. Figure 7 shows for each input type, the total number [illegible]nputs, and the number analyzed correctly and incorrectly.

CONVERSATIONAL ANALYSIS

INPUT	TOTAL	# CORRECT	# INCORRECT	% CORRECT
LEGAL	295	293	2	99.3
ILLEGAL	10	8	2	80.0
FORCED	71	71	0	100.0
TOTAL	376	372	4	98.9

Summary of Conversational System Performance

Figure 7

In addition it shows the percent of correct analyses associated with this operation. These results indicate a very high level of performance for the system. Not only does it handle valid inputs successfully, but it is also able to detect invalid inputs and treat them properly. The total number of inputs shown in Figure 7 is actually greater than the total number of input lines. This is because several inputs may be compounded onto a single line.

B) User Performance

The measures of user performance are necessarily more subjective than those of system performance. However, these results can provide useful information into the overall validity of this type of approach to a conversational implementation.

For each user, at least two dialogues are conducted; one with the user having a minimum of system knowledge, and one where he has more instruction and previous experience. On the first try, every user responded properly to the initial system [illegible]ion and was able to turn on the guide facility. Then using the guide facility, all but one user was able to successfully complete a simple retrieval process. The one exception did not understand the use of the word "default". After this was explained, the operation progressed normally. In general, all users were able to respond properly to the guide questions. The only major problem occurred at the end of the guided dialogue where the process is recycled and started again. It was not obvious to the user at this point, how he could sign off. But most users knew enough to request a tutorial which then explicitly displayed the available options; SIGNOFF being one of them. An example of this situation appears in Figure 6A. A slight modification of the final guide process can rectify this.

Having been guided through retrieval operation supplies the user with a great deal of insight into the use of the system. Using this experience and a small amount of added instruction to fill in any areas not touched by the guide facility, the user next attempts a normal (unguided) dialogue. All of the users tested were able to conduct a reasonable dialogue without outside help. A few of the users who had previous information retrieval experience were able to perform a highly competent retrieval after only a single introductory guided process. Of course nearly all of the users became stuck at some point and had to request a tutorial. Of the 32 tutorial calls made by all users, all but one supplied the information necessary for the user to continue. In some cases where the user received the master status tutorial, the single message answered [illegible] of the user's questions. He was then able to continue by making several references back to the same message. The one situation in which the tutorial did not help occurred when a user requested a tutorial during a guide process. Since the guide facility operates by generating successive tutorial messages, the user's request resulted in a repeat of the previously printed message. Thus the tutorial presented no new information. The user, however, was able to extricate himself by requesting a default option. In all the dialogues there was no case in which a user was forced to stop because he became hopelessly lost.

At the conclusion of each user dialogue he is asked his opinion of the system. The reaction of nearly all the users was favorable. They found the system both simple to learn and use. The tutorial facility is very well received, especially the convention of printing the appropriate tutorial in response to an erroneous input. Most of the critical comments center around revision in the wording of the various messages. A few of these messages are felt to be insufficiently clear to a new user. One user suggested that tutorials not only explain their options but also provide some samples of appropriate valid inputs. This comment, however, appears to be based on user timidity more than anything else. Unlike others, this user did not fully appreciate the natural language capabilities of the system and was afraid of submitting an erroneous input. He therefore wanted the sample input as a highly structured quideline for his input. But because of the ability of the system to treat natural language, such guidelines are unnecessary.

The overall feeling of the user is that the system provides an easy to use yet sufficiently rigorous conversational information retrieval facility. In addition the control conversational dialogue can be performed at each user's particular level of competence.

C) Timing

No analysis of a potential on-line system is complete without saying something about processing time. The current conversational program is written in FORTRAN and contains a great deal of diagnostic processing and output, as well as other debugging aids. It might therefore be considered that the timing statistics for the program would be somewhat worse than could be achieved using more efficient production programming techniques. However, these results do give a general idea of the processing speed. The timing of each operation varies from about 50 to 150 milliseconds. The complete set of 376 operations is performed in 37.057 seconds or about 0.1 second per input operation. When considering an actual console user, a rather conservative estimate for the average time between inputs (that is the time between end of input signals) is 10 seconds. In practice this average is probably higher. Thus at the rate of 10 conversational operations per second, the current system could adequately support a network of 100 consoles and supply one second or better of response time. Even with the inefficient code and conservative estimates, this is clearly within practical limits.

6. Future Extensions

There are a number of areas for future study with respect to the conversational system. First is a user storage facility. With this capability a user could store various aspects of his dialogue, such as queries or retrieved documents, for future use. In addition, a user could store parameters which would be automatically set at sign-on time. This would eliminate the need to specify the parameters each time he used the system. In addition the system can keep various statistics about its own performance which are valuable in evaluating and improving the system.

Carrying the storage capability one step further, the conversational system could be equipped with a learning subsystem. A user could then specify his own notation along with more conventionally stated equivalents. The system would then learn the user's special requirements. In this way a user could tailor the conversational system to his exact needs and conventions. The learning process could also be used in the treatment of erroneous inputs. This is shown in the sample script below. The user erroneously requests a nonexistent "BOOL" correlation. The system notifies him of his error and requests clarification and whether the incorrect input should be learned. After answering affirmatively, the user may then use "overlap" or "bool" interchangeably.

U: BOOL CORRELATION
INCORRECT CORRELATION, PLEASE CLARIFY AND
INDICATE IF INPUT SHOULD BE LEARNED.

U: YES, OVERLAP
UNDERSTOOD; BOOL = OVERLAP

Thus the learning process provides a way of meeting the particular needs of each individual user.

Some further work must also be done with respect to user terminals. Currently the most popular on-line communcation device is the teletype console. These are easy to use and relatively inexpensive. The most serious drawback is their slow output speed. A fairly simple tutorial may take 30 seconds or more to print. This can frustrate the user and needlessly tie up the terminal. Another type of terminal is based on a cathode ray tube (CRT). These permit almost instantaneous display of messages. In addition, part of the screen may be devoted to a prompting area. In this way the user always knows where he is in his dialogue and what options are currently available. Some CRT units have a light pen which allows selection of options by merely pointing the pen at the name of the desired option on the screen. However, there are several problems with CRT displays. First, the added hardware needed to drive a CRT makes them very expensive. Some work is being done by Bitzer [1] on the design of an inexpensive visual display unit which uses a plasma screen and slide projector. However, these are not yet commercially available. Alse the CRT produces no hard copy. A user might thus have to copy a long list of document numbers from the screen. The solution to this may be supplied by devices which contain both a visual and a hard copy facility. The user conducts his dialogue on the CRT. Whenever he receives something he wants saved, he indicates the appropriate subset of the script which is then printed. Such a device is currently being used experiemntally by the RIQS System at Northwestern University [13].

Another area for future study is the manner in which documents are displayed to the user. SMART and a number of other systems normally display only the document number. At best document numbers provide minimal information about the document's content. It might be better to store document titles or even abstracts on-line so that they may be seen by the user. This could be done best using a high capacity, low speed peripheral storage device. However, the expense of the dedicated storage device along with the prospect of having the terminal tied up printing abstracts, may make this technique uneconomical. Another possibility is to store document abstracts on microfiche. A set of microfiche and a reader would be supplied at each terminal station. The user would get a list of document numbers from the information retrieval system and then look them up off-line at the reader. Not only is the physical equipment for this cheaper than an on-line file, but also the fact that the scanning of abstracts is done off-line frees up the terminal for more useful work.

The fourth and probably most significant area for future development is the anl. sis of the conversational user. It is from this type of study that will come significant advances in tailoring systems to the actual needs of the system user.

7. Conclusion

Conversational information processing has many advantages over conventional batch methods. In this study [illegible] shown that it is quite reasonable to conduct conver[illegible] information retrieval in a natural language framework. Furthermore the template analysis process proves to be a useful technique not only for handling the natural language input to a conversational system, but it can take care of the bookeeping as well. The conversational system implemented using these techniques is shown by actual user experimentation to provide an excellent communication medium between man and machine.

References

[1] Alpert, D., and D.L. Bitzer, Advances in Computer-based Education, Science, Vol. 167 (March 1970).

[2] Atherton, P., and R.R. Freeman, AUDACIOUS, AIP Report, AIP/UDC 7, April 1968.

[3] Berezner, S.C., H.C. Carney, J.A. Craig and C.R. Longyear, DEACON: Direct English Access anc Control, General Electric Co., Proceedings FJEC, Santa Barbara, California, 1966.

[4] Bergman, S., W. Franks, E. Rubinoff and M. Rubinoff, Experimental Evaluation of Information Retrieval through a Teletypewriter, CACM, Vol. 11, No. 9 (September 1968).

[5] Bobrow, D.G., Natural Language Input for a Computer Problemsolving System, in Semantic Information Processing, M. Minsky, Ed., MIT Press, Cambridge, Mass., 1968.

[6] Bolt, R.H., Computer-assisted Socratic Instruction, in Conversational Computers, W.D. Orr, Ed., John Wiley and Sons, Inc., New York 1968.

[7] Curtice, R.M., and P.E. Jones, An Operational Interactive Retrieval System, Arthur D. Little, Inc., 1969.

[8] Dimsdale, B., and B.G. Lamson, A Natural Language Information Retrieval System, Proc. of IEEE, Vol. 54, No. 12 (December 1966).

[9] Halpern, M., Foundations of the Case for Natural Language Programming, IEEE Spectrum, Vol. 4, No. 3, March 1967.

[10] IBM Systems/360 Document Processing System, Applications Description, IBM, 1967.

[11] Ide, E.C., Relevance Feedback in an Automatic Document Retrieval System, Information Storage and Retrieval, Report No. ISR-15 to the National Science Foundation, Cornell University, 1968.

[12] Kellogg, C.H., A Natural Language Compiler for On-line Data Management, AFIPS Conference Proceedings, Vol. 33, Proc. AFIPS 1968 Fall Joint Computer Conf., Vol. 33, Thompson Book Co., Washington, D.C.

[13] Krulee G., and B. Mittman, Computer-based Information Systems for University Research and Teaching, Northwestern University, Evanston, Illinois, 1969.

References (contd.)

[14] Maceyak, J., A Question-answering Language for a SMART type Data Base, May 1968.

[15] Mathews, W.C., TIP Reference Manual, Technical Information Program, The Libraries, MIT, Cambridge, Mass., 1968.

[16] Meister, D., and D.J. Sullivan, Evaluation of User Reactions to a Prototype On-line Information Retrieval System (RECON), Appendix RECON User's manual, Report NASA-CR 918, Prepared by Bunker-Romo Corporation, Conoga Park, California.

[17] Moyne, J.A., PROTO-RELADES: A Restrictive Natural Language System, IBM, 1967.

[18] Moyne, J.A., A Progress Report on the Use of English in Information Retrieval, IBM Corp., Federal Systems Center, Gaithersburg, Maryland, June 1969.

[19] Moyne, J.A., Information Retrieval and Natural Language, IBM Corp., Federal Systems Center, Gaithersburg, Maryland, June 1969.

[20] Orr W.D., (Ed.), Conversational Computers, John Wiley and Sons, Inc., New York, 1968.

[21] Parker, E., SPIRES User Manual, Stanford Physics Information Retrieval System, Institute of Communications Research, Stanford University, Palo Alto, California.

[22] Reitman, W., R.B. Roberts, R.W. Sauvain, and D.D. Wheeler, AUTONOTE: A Personal Information Storage and Retrieval System, Mental Health Research Institute Communication #248 and Information Processing Working Paper #12, University of Michigan, Ann Arbor, Michigan, 1969.

[23] Rubinoff, M., S. Bergman, H. Cautin, and F. Rapp, Easy English, A Language for Information Retrieval Through a Remote Typewriter Console, CACM, Vol. 11, No. 10 (October 1968).

[24] Salton, G., Automatic Information Organization and Retrieval, McGraw Hill, New York 1968.

[25] Salton, G., Interactive Information Retrieval, (Unpublished).

[26] Silvern, L.C., CAI in an Expanding Universe of Educational Methodology, in Conversational Computers, W.D. Orr, Ed., John Wiley and Sons, Inc., New York 1968.

References (Contd.)

[27] Simmons, R.F., Synthex, in Conversational Computers, W.D. Orr, Ed., John Wiley and Sons, Inc., New York, 1968.

[28] Simmons, R.F., Natural Language Question-answer Systems: 1969, CACM, Vol. 13, No. 1 (January 1969).

[29] Summit, R.F., DIALOG II Users Manual, Information Science Electronic Science Lab., Lockheed Palo Alto Research Lab., Lockheed Missiles and Space Co.

[30] Thompson, F.B., DEACON Type Query Systems, in Conversational Computers, W.D. Orr, Ed., John Wiley and Sons, Inc., New York, 1968.

[31] Weiss, S.F., A Template Approach to Natural Language Analysis for Information Retrieval, Ph.D. Thesis, Department of Computer Science, Cornell University, Ithaca, New York, 1970.

[32] Weiss, S.F., Template Analysis and its Application to Natural Language Processing, Information Storage and Retrieval, Report No. ISR-16 to the National Science Foundation, Cornell University, 1969.

[33] Weizenbaum, J., Contextual Understanding by Computers, CACM, Vol. 10, No. 8 (August 1967).

[34] Weizenbaum, J., ELIZA — A Computer Program for the Study of Natural Language Communications Between Man and Machine, CACM, Vol. 9, No. 1 (January 1969).

[35] Williamson, R., A Prototype Document Retrieval System. (Unpublished).