I. Design of a Revised On-Line Information Retrieval System

M. E. Lesk

1. Introduction

Experience gained with the existing SMART retrieval system under experimental conditions justifies an attempt to expand the automatic search and analysis methodology to operational environments. A revised SMART system is proposed for this purpose, and a practical design for a flexible, powerful retrieval system on the IBM 360 series computer is accordingly submitted.

The goals for the new SMART system are threefold. First, to investigate retrieval procedures for systems of operational size. This includes analysis and search procedures suitable for large systems, file organizations, dictionary handling, and the provision of a reasonable number of options and output procedures to permit a study of the final system and its evaluation. Second, to investigate user interaction with real-time systems, since this may be expected to be a major requirement for future information retrieval systems. User interaction, of course, must be based on a system with console facilities and real-time speed. Third, to provide the programming framework for a truly operational system with real users.

In addition to the proposed design, it will be necessary to provide only a very large data base and additional consoles in order to generate a complete, operational retrieval center.

The specific capabilities demanded from the proposed system include the ability to deal with collections of many thousands of documents. Eventually, collections of up to 250,000 documents may be processed with the programs. However, the basic storage options chosen for a first implementation accommodate collections of about 25,000 documents. A maximum of five on-line consoles is initially anticipated. The first system may be expected to use one disk pack as a bulk storage medium. Neither data cell nor drums are needed initially. The system must be flexible in its use of memory, so as to run on almost any configuration. Thus, storage space must be allocated efficiently, for allocations ranging from 50,000 bytes upward to the complete memory. Most of the space allocation is done automatically by the supervisor. It is believed that 50K bytes would still permit a 25,000 document system to run economically.

Later versions of the system will use a disk pack plus a data cell for storage, and operate with a larger number of consoles (up to 50). Overall organization may not change, however. In addition, the

updating of the document collection and the dictionaries can proceed simultaneously with the processing of requests; the batch-processing type "background" work is handled efficiently and forms a useful adjunct to the foreground request work.

The file organization of the system strikes a balance between ease of updating and efficiency of running. No massive sorts or merges are required to prepare or update files, nor are special "new material" files kept. These requirements minimize programming and overhead time and avoid inconvenience for the system operators.

The user interaction process requires rapid response times and flexible commands. The system design goal was a response time of ten seconds. A variety of analysis processes can be specified by the user as in the present system, and the flexible system design enables easy introduction of new procedures. In addition, provision is made for user-oriented system responses in which the processing algorithms depend on the individual user's past history. This user adaptation feature offers the equivalent of an individual retrieval system for every user.

The third requirement, that the system serve as the programming framework for an operation center, implies

that the system must be designed to accept a variety of input documents, in several languages, and of several types. The system handles multi-lingual input by a mult-lingual thesaurus, and uses a flexible document representation well suited to the problems of an operational system. The ease of updating and the user adaption features also simplify the conversion to actual documentation operations with real users.

The most serious demand placed on the system is the flexibility required by its dual function either as a batch processing system or as a real-time request answering system. Both systems are necessary because the task of adding new documents to a retrieval data base must remain a batch task, while request answering is inherently a real-time time-sharing operation. Furthermore, in the early stages of system development, batch processing must be relied upon if interactive consoles should not be available on time.

2. Supervisor Organization

The problem of designing a system to meet the criteria specified in the Introduction, including in particular the one requiring that the system be used both as the framework for operational retrieval and for experimental work, demands not only fast, convenient algorithms,

but also a supervisory organization and scheduling methods suitable for both purposes. In addition, since the system is to be improved as it is used, the supervisor must allow the simple introduction of new algorithms and procedures into the system.

Ideally, the scheduling problem might be solved by turning it over to an operating system monitor of the time-sharing type. In practice, such monitors are not available with systems of the size and capacity needed for retrieval purposes, and their overhead requirements are such as to make it impossible to perform efficient production runs. For this reason, a small-scale scheduler is provided as the heart of the new system. This scheduler is designed to

a) reconcile the needs for batch processing and console interaction;

b) provide for easy substitution of programs and data files and data processing steps in the retrieval procedures;

c) provide efficient machine utilization.

It is assumed that the system monitor used is such that control is given to the SMART programs only once, and that the computer cannot be relinquished with the expectation of getting control back within a short period of time.

The supervisor operates by maintaining two basic files, a list of processing tasks and a list of data items needed for these tasks. A dictionary which relates retrieval procedures to the specific tasks and data items needed is used to make entries into this table. For example, the addition of an input text to the main data file might proceed as follows:

a) user types: "addtext 5B465"

b) program looks up "addtext" in procedure dictionary, breaks it into tasks:

   i) accept input of text from console;
   ii) perform dictionary lookup;
   iii) print out words not found;
   iv) compare text with existing file arrangement;
   v) place text in file.

c) for each of these steps, which are entered in the processing task list, certain corresponding items are placed in the data list. In the example, the data list might appear as follows:

   i) ---
   ii) dictionary sections; input text
   iii) list of words not found
   iv) file arrangement keys
   v) main file section (generated in step iv)

d) the supervisor could now proceed to execute the appropriate initial steps.

Note that a processing task may add to the processing task and data lists; it may also generate data files needed by the later steps. This permits the execution of powerful procedures. For example, the program might include criteria specifying that if only 2 words, or fewer, are not in the dictionary, these are printed out, but nothing else is done unless requested by the user; if more than 2 words are not found, the user is asked to supply synonyms or alternatives for the unknown words. The existence of a "dictionary" explaining the system commands, and the ability of the programs under execution to add or delete from the command structure, permit easy alteration of the retrieval procedures with full facility for the use of sophisticated, highly-optimized retrieval algorithms.

Normally, the supervisor controls several different retrieval procedures under way at once, each of which is represented by a set of entries in the task and data lists. In order to determine what is to be done next, the supervisor computes a parameter called "priority" for each processing task, and a parameter called "accessibility" for each data item needed. The priority of a task depends

on whether this task originated with a user at a console or with a background job; whether the user in question possesses any special priority relative to other users; and how long the task has been waiting.

The "accessibility" of the data items is somewhat more difficult to define. The supervisor maintains lists of the location of all needed items in terms of tape drive and disk addresses. Programs within the system are only allowed to use symbolic references to data files. This permits the easy adaptation of the system to operations with one disk or two disks and to changes in the unit on which a tape is normally mounted. The symbolic addressing also generally eases the job of an operator of a multi-programmed computer for which it is not always known what tape or disk drives will be empty at any given time. As items are brought into core memory, this fact is also noted in the supervisor tables. When a request for a data item is made through the data list, the supervisor determines how "accessible" the item in question is. The accessibility may be thought of as the ease of bringing the item into memory. The accessibility of an item in core is infinite. The accessibility of an item on a tape or disk drive is given some number varying inversely with the time needed to bring the item into core which in the final system could actually be computed using the known tape or disk

addresses of the item and the present position of the read head. The accessibility of an item on a tape or disk pack not mounted on the computer is some very low number; and the accessibility of an item which is to be generated by a processing step not yet completed is zero (representing complete inaccessibility).

The supervisor now computes for each task a parameter called "importance", representing a weighted combination of priority and accessibility. The weighting is adjusted so that in a primarily batch-oriented system, accessibility is much more important than priority, while in a console-oriented system, priority is more important than accessibility. Whenever there are conflicting demands on either central processing unit or on any I-0 unit, preference is given to the task with the highest importance.

After a task is completed, it naturally is deleted from the task file. When an I-0 job is completed, any transfer into the computer is noted in the supervisor's table of data locations. This table includes a note of the various memory locations used, (memory being divided into standard-sized buffers) and of the importance of the locations for the given task. Any transfer out of the computer is also noted, indirectly, by the elimination of any importance that was attached to the core area involved, and therefore, by the release of the area for other use.

The core buffers are also used on the basis of "importance"; a more important task in need of buffer space may use core space intended for a less important task. For example, if a section of document vectors is brought into memory for a search on a background query, and a high-priority console query is suddenly submitted which demands buffer space for the dictionary, this space would be taken from the space used by the background jobs. When the supervisor later returns to the background job, the data will be reaccessed from bulk memory.

A flowchart of the supervisor is shown in Fig. 1. Control is passed to the supervisor whenever a task is finished, when an I-0 unit has finished operation, or whenever a console requests attention. At all such times, the supervisor checks through the processing task file and the data needed file. It makes the appropriate additions or deletions, recomputes the priorities and accessibilities of all tasks, and assigns all available units to the task of highest importance presently in the files.

Normally, both the processing task and the data files needed will have entries, and the supervisor will have no difficulties in assigning the tasks. Occasionally, however, one or both files may be empty. If both files are empty, this implies that all work is finished, and unless further console input is expected, the job is terminated. If the

processing file contains no executable jobs, but the I-0 equipment is working, this implies that all work is waiting for input-output, and the supervisor idles until the necessary information is in core. If the processing file contains tasks, but the I-0 equipment is idle, this implies that the most important task is not I-0 limited. Normally, the supervisor would continue to find additional data necessary for less important tasks, but occasionally, all buffers may be full. In this case, the I-0 equipment waits until some buffers are released by the processing system.

Note that as long as importance is mostly determined by accessibility, all material in core is used before extra disk accesses are made, even if high priority tasks should be slowed down. This permits the efficient batch processing of lookups and searches. On the other hand, if priority is set as the determining influence, the most important request is processed first, even at the expense of extra disk accesses. Thus, by changing the relative weights of priority and accessibility, the system changes from batch-oriented to console-oriented.

3. System Procedures

A variety of procedures for retrieval are needed in the operating system. The procedures outlined here are not intended to be exhaustive; but they should provide a

flexible, useful retrieval system. All procedures need not be implemented at once, of course; it is relatively easy to expand the system with time. Only the basic input, lookup, and searching parts need be programmed initially.

In order to facilitate programming, the data structures used in the system should be simple, easily read and written, and few in number. The major files are:

a) the input text, in original English form. This should be stored, if space is at a premium, in the digram encipherment described in the appendix;

b) input text in foreign languages, stored in formats appropriate for each language;

c) the word stem dictionary for English (and separate dictionaries for foreign languages); these can also be stored in the digram encipherment if space is needed;

d) automatic thesauri, referred to the stem dictionaries rather than to word lists;

e) document vectors for looked-up documents;

f) cluster directories for the collection.

Each file is further described later in this report. The major procedures used by the system are:

a) request and text input;

b) request and text lookup, stem dictionary;

c) automatic thesaurus processing;

d) phrase processing;

e) hierarchical operations;

f) concept vector formation and storage;

g) search of document collections or subcollections (request-document comparison);

h) clustering of document collections;

i) relevance feedback;

j) dictionary displays;

k) citation searching;

l) class information (language, date, type of article, etc.);

m) selective dissemination of information (SDI);

n) user information files.

These operations are discussed in the following sections.

A) Request and Text Input

This involves no special difficulties. The system merely accepts input strings and stores them. Text may be input from either a high-speed unit or a console. In accordance with general system formats, console input would be read into high speed memory and also stored on a slow-speed unit. This permits the user to read out his original question if after a series of modifications he wishes to resume from scratch. It also protects the user against a higher-priority user taking over high-speed memory; and it permits the system managers, at the end of the day, to see what queries have been submitted.

Actual input would involve either keypunching followed by card-to-tape on offline equipment, or transmission, from a teletype or other console unit. If extremely high volumes of input are needed (as is true when acquiring the data base for an operational system), optical character recognition equipment could be considered. This would be economical only if input costs were at least in the $50,000-$100,000 range.

As a subsidiary part of the input process, a good text editor should be available. Such editors are essential in handling any large quantity of text. A new one should be written if not available from program libraries.

The supervisor list entries for input commands consist of the following:

	Command	Task List	Data List	I-0 Unit	Task
i)	console input	read console	--	disk storage core table	writer query enter query
ii)	tape input	read tape	input text tape	core table	enter document

The reading of queries previously placed on the disk (to be processed in batching mode) follows the same pattern as tape input except that a disk file is substituted for the input tape.

B) Request and Text Lookup

The lookup procedure is the first of the two major operations of the retrieval system (the other being the collection search). The basic properties cf the lookup

procedure must be:

a) high speed

b) minimal core requirements (the entire dictionary is too big to fit into memory

c) ease of extension (new words must be added constantly.

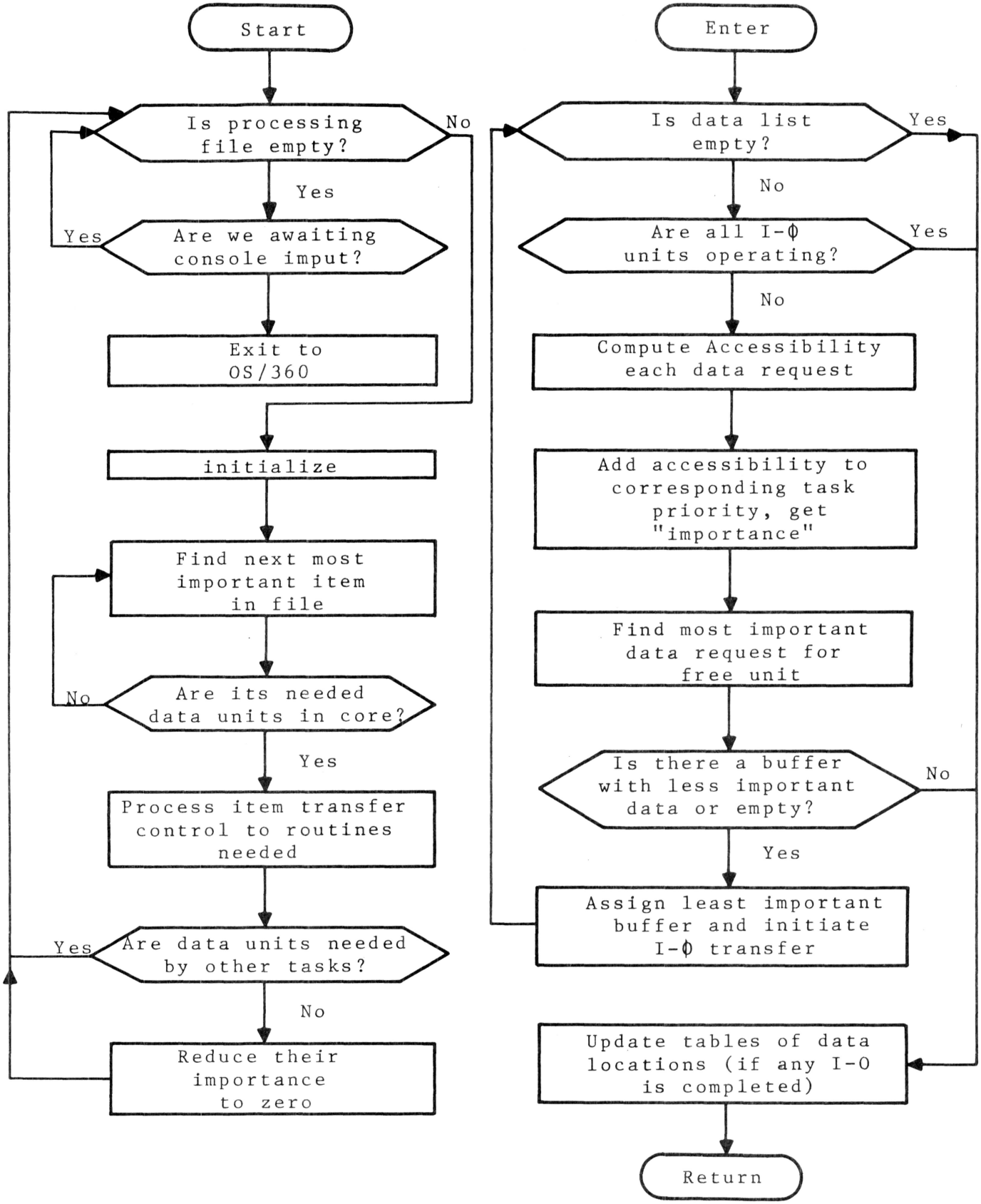
To obtain high speed, it is important that disk accesses be minimized, because the random-access time of a disk is about 100 msec, compared with microseconds for core accesses. A solution to this problem can be derived from Fig. 2, which represents a familiar frequency vs. total occurrences chart for a sample of English text. One percent of the distinct words represent 46% of all word occurrences. This permits a high-speed lookup by keeping a small, high-speed dictionary in memory and putting all low-frequency words out on the disk pack.

Normally, dictionary lookups are based on lists of words stored in an alphabetic representation. This method suffers from several disadvantages, namely:

a) inserting a word into alphabetic order requires a great deal of data handling;

b) if the words are not grouped by length, space is wasted; if they are, the look-up for suffixed words is complicated and slow;

c) the high and low-frequency dictionaries operate with completely different search algorithms, and the time spent searching the high-frequency dictionary is

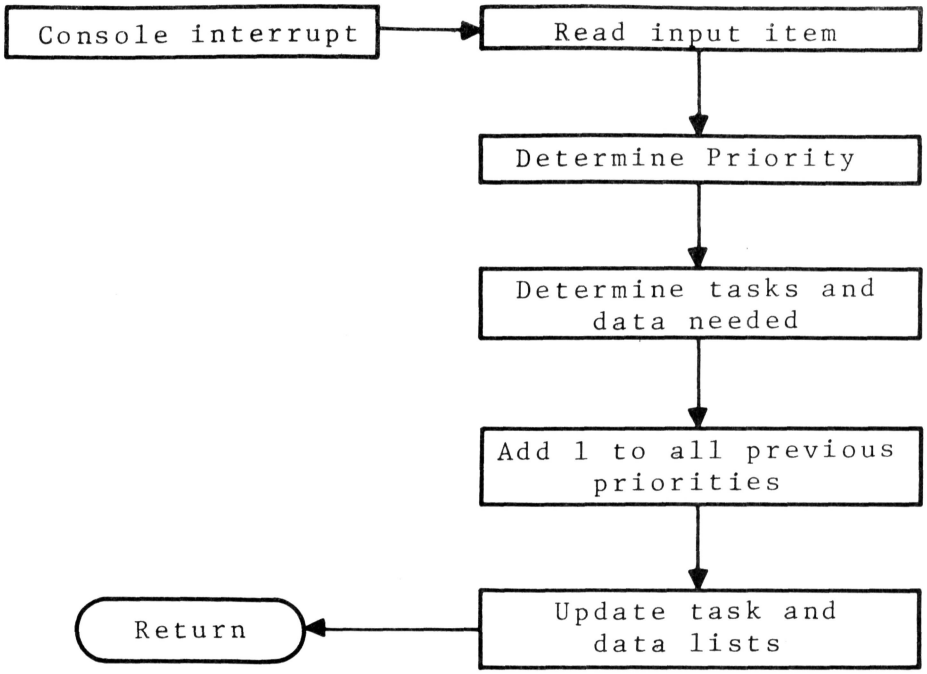


a) Main Routine

b) I-O Routine

Supervisor Organization

Fig. 1



c) Input Section

Input	Task	Data List	Output
English request	Lookup	Dictionary	Words not found, stem vector
Stem vector	Thesaurus expansion	Thesaurus	Concept vector
Concept vector	Centroid search	Centroid vectors	Promising clusters
Concept vector	Cluster searches	Document vectors	Answers

d) Sample Query Processing

Supervisor Organization

Fig. 1 (contd.)

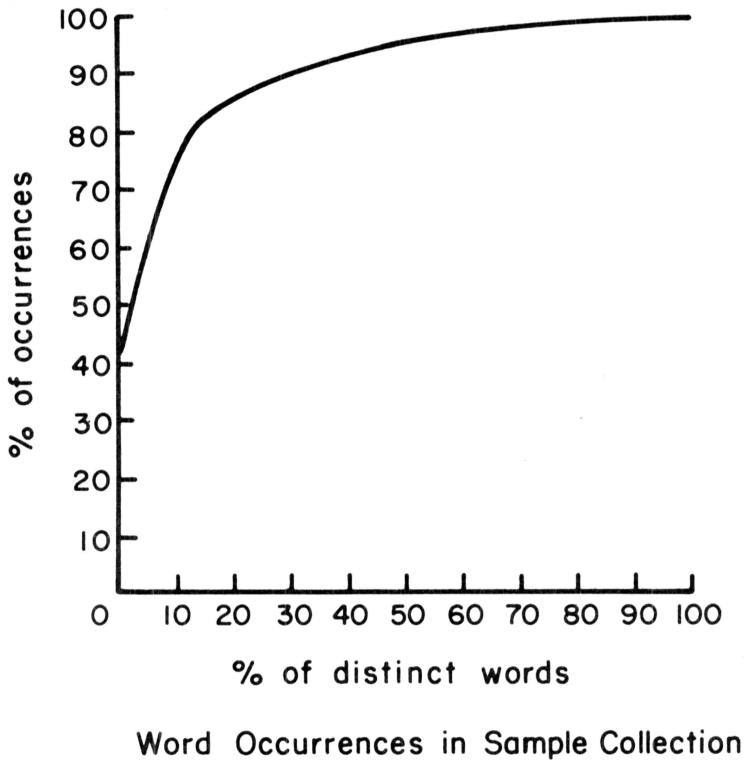

100
90
80
70
60
50
40
30
20
10
0
10 20 30 40 50 60 70 80 90 100
% of occurrences
% of distinct words
Word Occurrences in Sample Collection

Fig. 2

wasted when the word is not there;

d) complicated updating problems are created unless concept numbers independent of the words are stored with them, thereby wasting more space.

This is not to say that a feasible lookup using alphabetic characters could not be devised; such a lookup would probably involve a high-frequency dictionary in core, stored by word length, with a low frequency dictionary on the disk, stored alphabetically with concept numbers. The digram encipherment of the appendix would be used to save space. Such a system, however, is believed to be inferior, for practical purposes, to a scatter storage scheme devised by R. Williamson and D. Murray, described elsewhere [2]. In this scheme, each word is "hashed" into an arbitrary 32-bit number in such a way that it is extremely unlikely that any two words will generate the same number. The actual probability is on the order of $N/2^{32}$, where N is the number of words in the dictionary. For collections of practical size, this implies that about 1 conflict is expected for a collection of 100,000 distinct words. This rate is well below other error rates associated with keypunching, data transmission, and so forth. The 32-bit hash is then divided into a "major" and a "minor" part. The major part is used to access a table kept in memory, which indicates the most

probable minor part for that major, and the location of a table of other minor parts. The hash scheme requires the following tables:

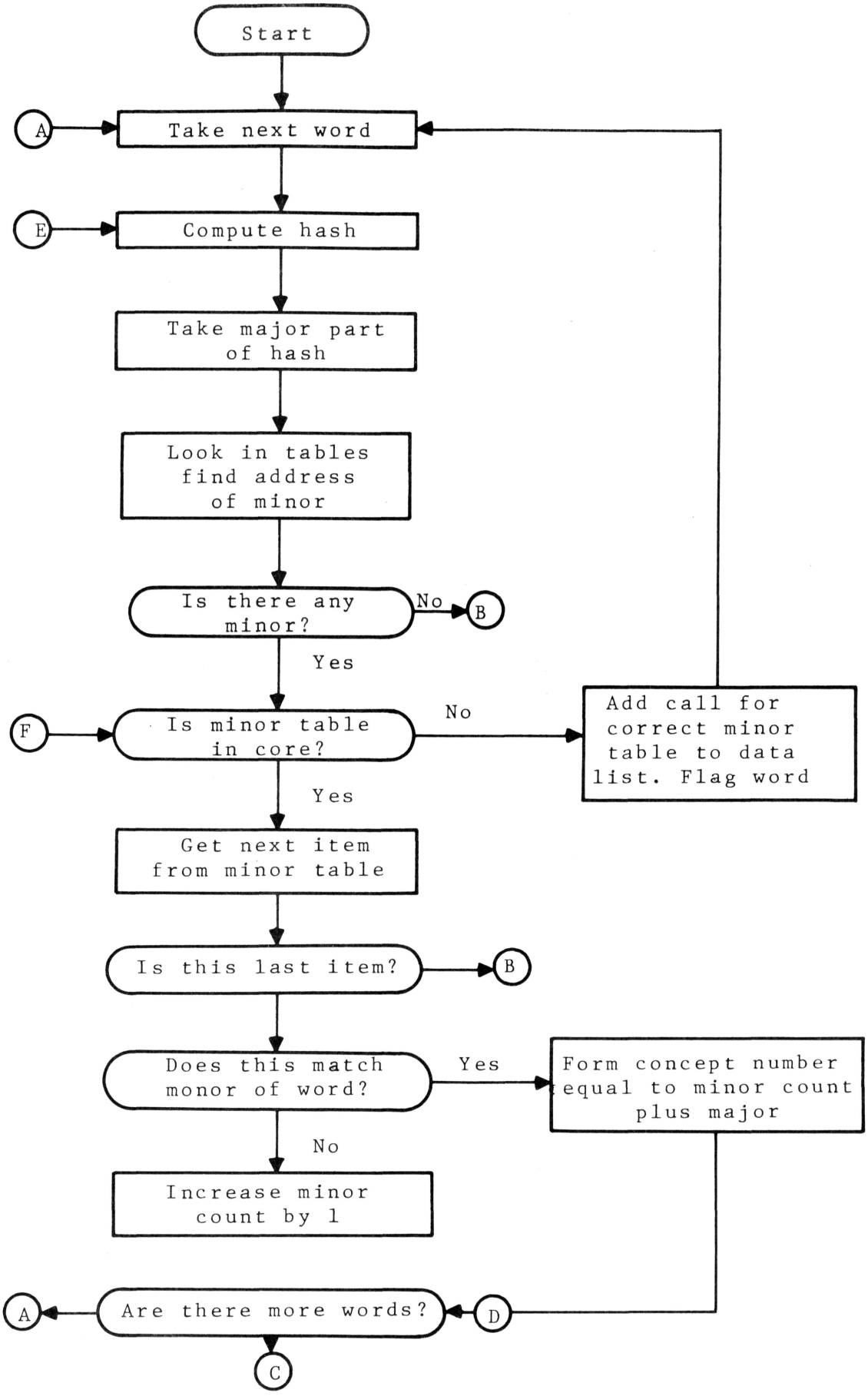
a) a table of initial minors and addresses for them; this table, constituting the high-frequency dictionary, must remain in core at all times;

b) various tables of minors and further addresses; these are ordered to permit the most frequent of the words still remaining on the low-frequency list to be kept in memory if extra space is available. Depending on the demands placed on the supervisor, some of these extra tables might thus still be placed in memory at all times.

This process offers a significant advantage over the alphabetic lookup. In the alphabetic lookup, with two distinct forms of the dictionary, no substantial speed is possible by bringing small portions of the low-frequency dictionary into core from the disk.

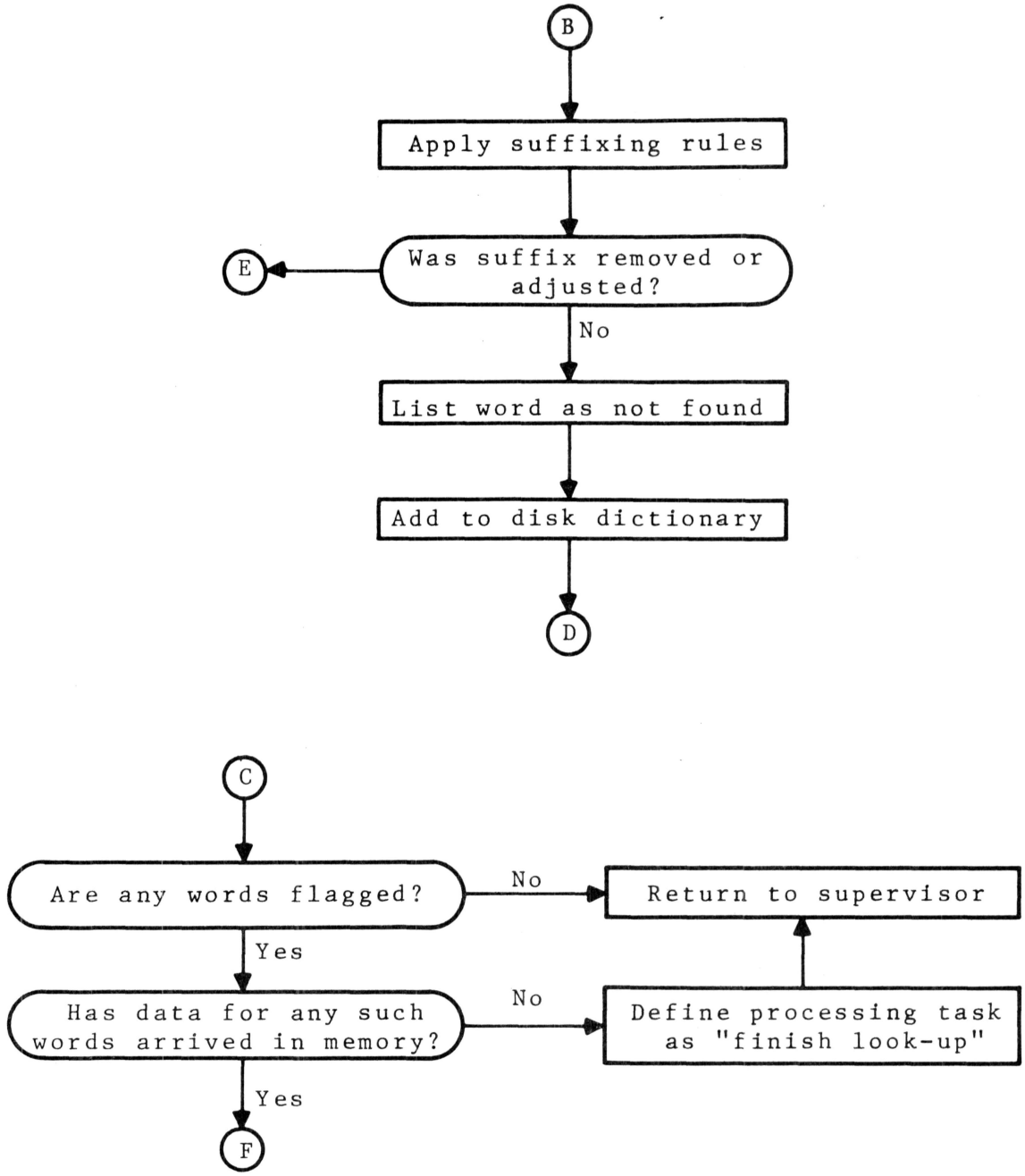
Trading memory for lookup time is very important in a system in which bulk lookups can be done at night with large amounts of memory, but small lookups must be done during the request searching, taking place during the day when there is competition for memory.

A glowchart of the lookup program is shown in Fig. 3. Each word is first searched in its original form. If this



Dictionary Look-up Routines

Fig. 3



Dictionary Look-up Routines

Fig. 3 (contd.)

fails to produce a dictionary entry, new searches are made as follows: the right end of the word is checked against a suffix list. If a suffix is removed, and the potential stem ends in a double consonant, that consonant is changed to a single consonant. If the potential stem ends with a single consonant, and the possible suffix begins with a vowel, the final "e" is added to the stem. If the potential stem ends in "i", it is changed to "y" (if the suffix begins with a vowel). This new stem, the "most probable" stem, is then searched. Should this fail, the search is repeated without the stem modification. If the stem is still not found, further suffixing is attempted and the searches continue.

Each search is performed by computing the hash for the word, followed by addressing a table to find the minor list corresponding to the major for the hash. Minor lists not in the dictionary blocks already in memory are accessed from the disk storage. Words which require disk fetches are temporarily bypassed by the processor until the necessary disk records are brought into high speed memory.

When a word cannot be found in the dictionary, two options are available:

a) the word can be typed on the console, where the user has the option of typing a substitute. This is the normal procedure for console input.

b) the word can be used as is, and entered into the dictionary. A note is made in a special file used by the system managers, giving them all the new words entered during this lookup. This list is then perused and the words are categorized as

   i) spelling errors; such words are removed from the dictionary and the original documents are fixed.

   ii) new useful words; these words are placed into their proper thesaurus categories and retained in the dictionary.

Each word retained in the dictionary is also stored in a file giving the English forms of the words preferably in digram form. These English forms of the words are used for thesaurus printouts, for the compilation of frequency lists, and so forth. They are arranged by hash concept numbers.

The operation of the hash procedure follows that designed by Murray [2]. Each hash is split into a thirteen-bit major and a twenty-five bit minor part. The thirteen bit majors are used to address a table of 8192 items, noting the beginning location of a table of minor parts for each major. The table of minor parts contains 25-bit minors, and flags indicating whether or not the word in question is common, and whether this minor part is the last entry in the minor table, or whether this 25-bit code is an address in a new block for the continuation of the list of minors. The minor

tables are distributed among blocks in such a way that the first block of minors contains all frequent words; except for compromises required by storage efficiency, the most frequent words of those remaining are stored in the next few blocks, and the least frequent words in the last blocks. This permits the system to trade memory space for lookup time.

Under normal circumstances, words are not checked against their English forms, since the table of English forms would be rather bulky. At periodic intervals, however, a list of words obtained from concordances of the input documents should be looked up and checked against the originals. This indicates whether any two different words were assigned the same concept number, and permits the insertion of an error flag at that minor table entry, thus catching and correcting the error in the future. If a dictionary of 100,000 words is considered (probably larger than the actual size needed for several years) a total storage requirement of: 16,384 bytes (13-bit major tables) and 400,000 bytes (25-bit minor tables plus flags) would be expected. The 400,000 byte table would be divided into twenty 20,000 byte blocks, each containing 5,000 different word stems. The system could operate with 36,000 bytes in memory permanently and one 20,000 byte storage area; alternatively, more memory could be utilized to speed up the lookup. In particular,

bulk lookups could be done using an LCS module (presumably at night). This would permit very fast lookups.

For comparison, if an English dictionary, averaging eight characters per word, were used, 800,000 bytes would be received in a word-length array, with concept numbers, using digram encipherment. This would produce a much slower search, because the high-frequency dictionary and the low-frequency dictionary would both have to be binary searched, and no way exists for arranging the low-frequency dictionary in frequency order. Furthermore, the dictionary would be difficult to update.

The time required to search the dictionary depends on the number of blocks which could be kept in memory at once and the total size of the dictionary. Imagine a dictionary of 25,000 words, representing 4 blocks. In the minimum space of 2 blocks, one block could be kept permanently in memory. This represents over 90% of the words in the average query, and leaves only about 3-5 words to be searched in the disk file. At 100 msec per disk access, about 0.3 to 0.5 seconds would be required to look up a request.

The output of the lookup consists of an ordered list of concept numbers and weights. The concept numbers could be stored with the ordinal number (the number of passes through minor tables) in the high-order bits, and the major part of the hash as the low-order bits. In this way, the most common

words would have the lowest concept numbers. This greatly simplifies such tasks as thesaurus expansion. The supervisor list entries for lookups are shown below.

Command	Task List	Data List	I-0 Units	Queues
lookup	compute hashes	major table hi-freq minors	disk disk	major table read in high-frequency minors
	search minors (enter needed in minors data list)	low-freq minors	disk	low-freq minors
	finish concept vector, write out		disk	write concept vector

C) Automatic Thesaurus Processing

Thesaurus processing makes use of simple numerical tables corresponding to the hash concept numbers, and giving thesaurus concept numbers for each hash (stem) concept number. Because the thesaurus contains many fewer concept numbers than the stem dictionary, a 16-bit concept number should be adequate, instead of the 16-bit plus ordinal number required by the hash stem concept. The initial table of thesaurus concept numbers, used for the first sequence of hash concepts, is simply an addressable table, used for the lowest (and most frequent) concept numbers. Since the highest concept numbers are more sparsely distributed, the lookup proceeds with a binary search rather

than by direct fetches. Thus, the size of the thesaurus, for 100,000 null concepts, would be:

a) a table for the first (for example) 2^{14} concepts, representing 32,000 bytes at 2 bytes per concept, resident in core.

b) a table for the remaining 84,000 concepts, representing only those concepts which exist and for which the stem concepts are not adequate, involving about 40,000 concepts at 6 bytes per concept, or 160,000 bytes, probably broken down into 8 blocks of 20,000 bytes each.

For stem concepts not placed into thesaurus categories, retrieval would be based entirely on the word stems. This permits the thesaurus to be built up gradually.

The speed of the thesaurus process would be high, since a greater fraction of the thesaurus is in memory than of the original dictionary. For a practical-sized thesaurus with 25,000 stems, it is not unlikely that the entire thesaurus could be kept in about 50K bytes of memory during the entire process, producing a lookup time per query (50 words) of about 500 usec. The high speed of the thesaurus look-up (around 10 microseconds per word, since about 90% or more of the words would require only one core access) permits an entire collection, of 50,000 abstracts to be processed in only about one minute. This implies that the thesaurus may be revised frequently, and all documents reprocessed as often

as is necessary, without difficulty. This feature alone provides a big advantage over manual systems in which it is often impossible to revise an indexing scheme once it has been used for a long period of time, because of the huge labor involved in indexing the backlog.

An alternative worth considering is the use of the thesaurus at search time, to expand documents and queries immediately before they are matched. This saves the disk space needed to store thesaurus vectors with the collection. However, it causes a shortage of core space at search time, when core space is in short supply, and it slows down the search process, which is not I-O limited at all times. Since it is inexpensive to process the entire collection, it is recommended that this be done in advance and the vectors stored.

The normal format of the thesaurus tables is a two-byte entry for each stem concept. This provides one thesaurus concept for a stem concept. If several thesaurus concepts are needed, a special flag is set and the 15-bit number is used to address a table in which the several concepts are listed. This economizes on space and eliminates restrictions on the number of thesaurus concepts that can be associated with one stem concept.

Foreign language material is processed using thesauri in which the same concepts are expressed by different words. Separate stem dictionaries are used for the foreign language material, and the foreign language

thesaurus concepts are defined in terms of appropriate stems. Each concept number, however, has approximately the same meaning as the corresponding concept in the English thesaurus.

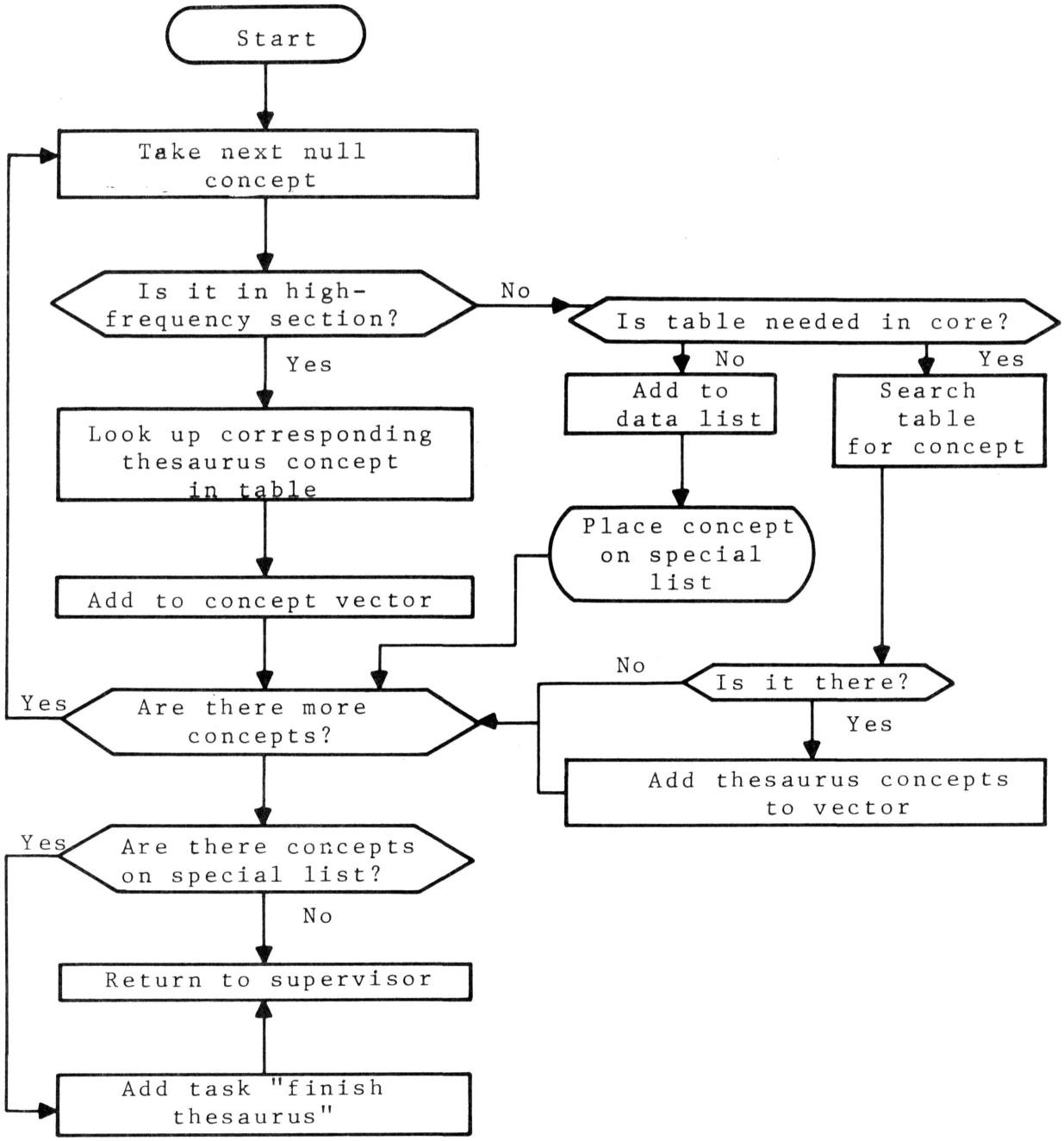
A flowchart of the thesaurus process is shown in Fig. 4. The task and data list entries are shown below:

Task List	Data List	I-0 Unit	Queue
thesaurus translation	thesaurus, high-freq. section	disk	thesaurus, high frequency section
	low-freq. sections as needed		
condense and sort; form vectors	write out vector		

D) Phrase Processing

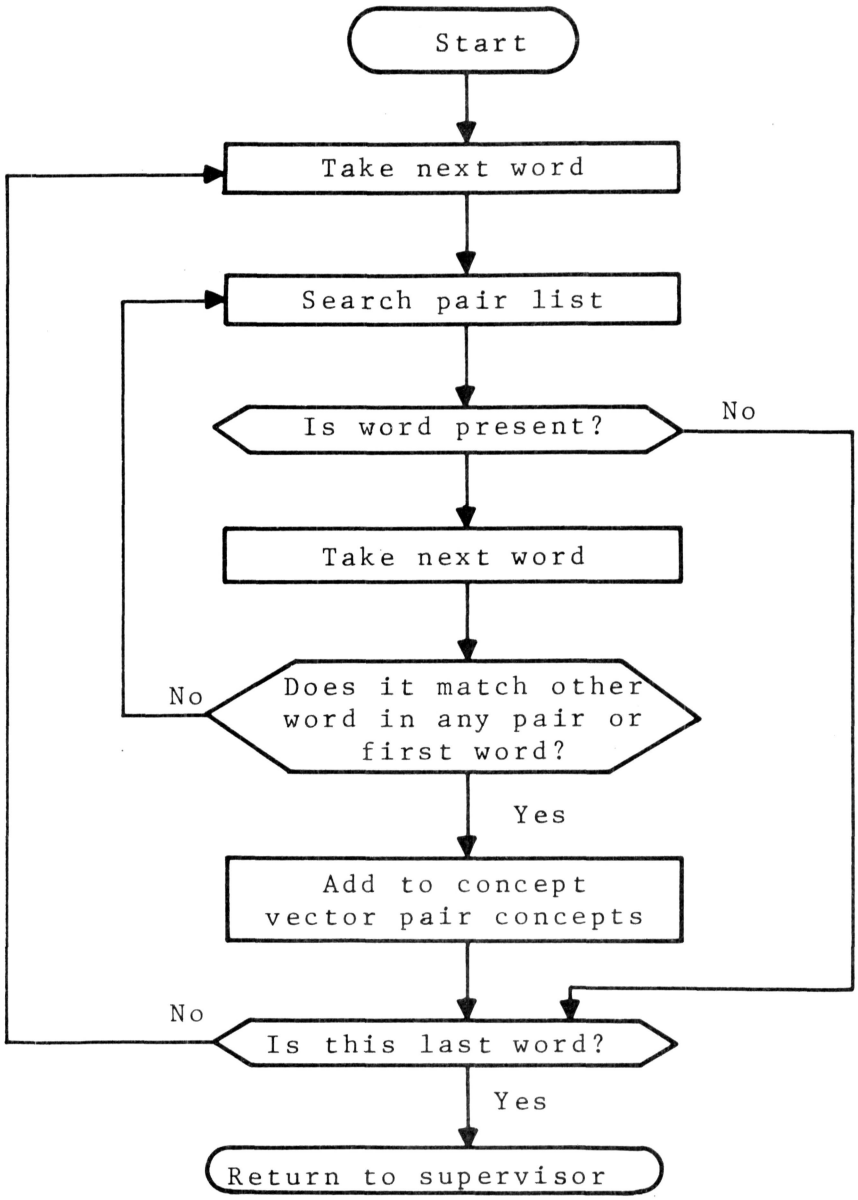
This step is not in the minimal system. Phrase processing could use an intermediate file, created by the stem lookup, which preserves the stem concepts in sentence order. A separate dictionary then lists the stem concepts used in phrases, and would be brought in to process the intermediate file. The resulting concepts could be added to the main concept vector.

The entries in the task and data lists are shown below. Possible flowcharts for the phrase processors are shown in Fig. 5. The first flowchart represents a phrase system that detects two adjoining word pairs, such as "information retrieval"



Thesaurus Program

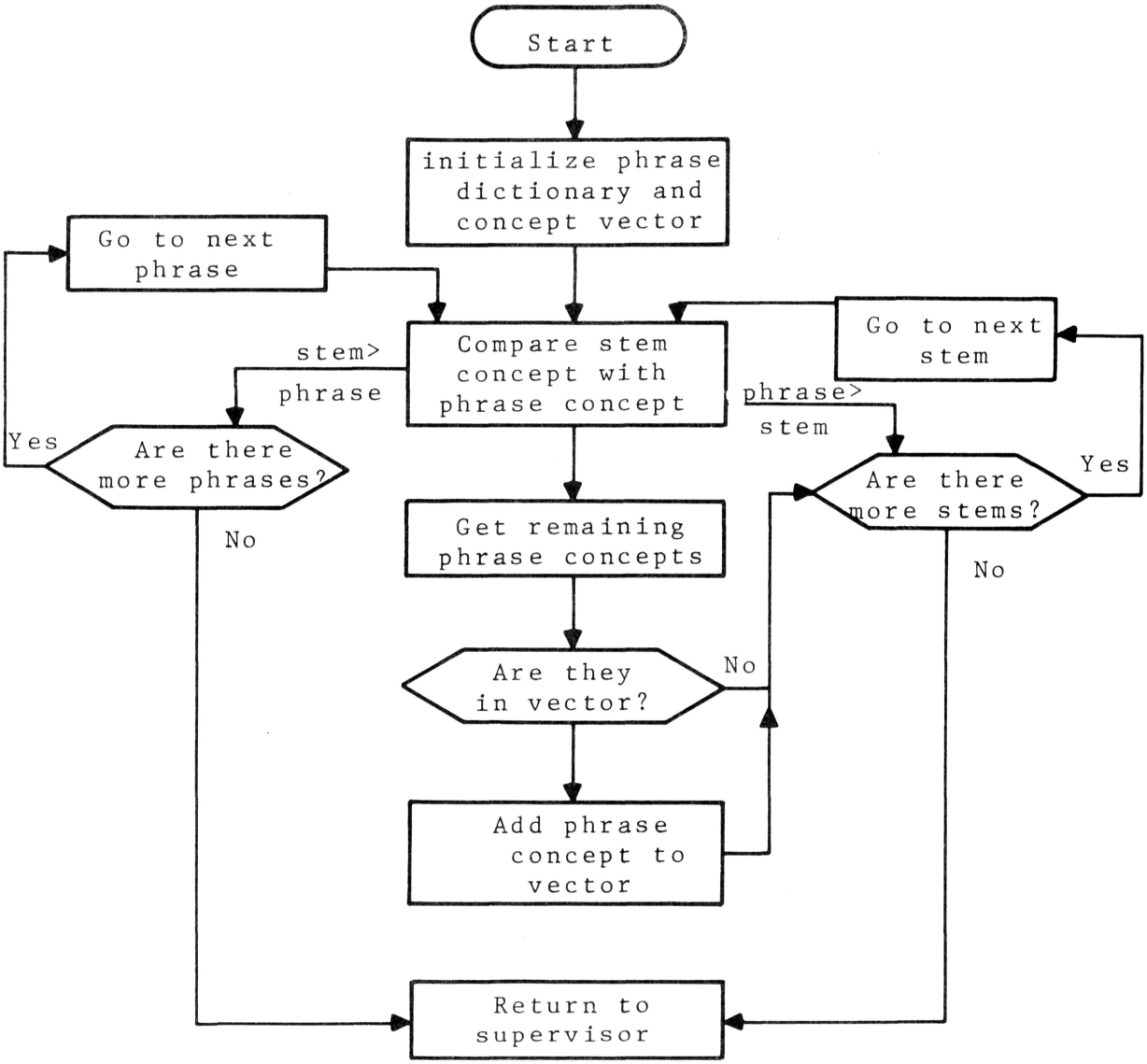
Fig. 4



a) Word Pair Phrase Generator

Phrase Generation

Fig. 5



b) Unlinked Phrase Searcher

Phrase Generator

Fig. 5 (contd.)

but nothing else. The second flowchart represents a phrase processor that searches for phrase components in a text. This processor would accept constructions such as "information and document retrieval". It does not use the intermediate, sentence ordered file, but only the ordinary stem vector.

Task List	Data List
phrase search	phrase dictionary document vectors
write new vector	document vector output

E) Hierarchical Processing

Previous experience with hierarchies has not led to great success, and therefore it is not recommended that a complete hierarchy be used [3]. Instead, an expansion corresponding to the normal expansion to "parents" in a full hierarchy is suggested. This option might be of some use in a retrieval operation for poorly performing queries. An example is shown in Fig. 6.

The operation of the hierarchy is very simple, because the hierarchy table merely contains the parent class of each concept, for a total of less than 2^{15} concepts. The hierarchy can thus be kept in memory. The thesaurus concepts are used in preference to the stem concepts, of course.

Query:	How are diodes and triodes used in IBM machines?
transformation	diode, triode } → vacuum tube IBM → computer
document matching improved query	"vacuum tube computer"

Use of Hierarchy

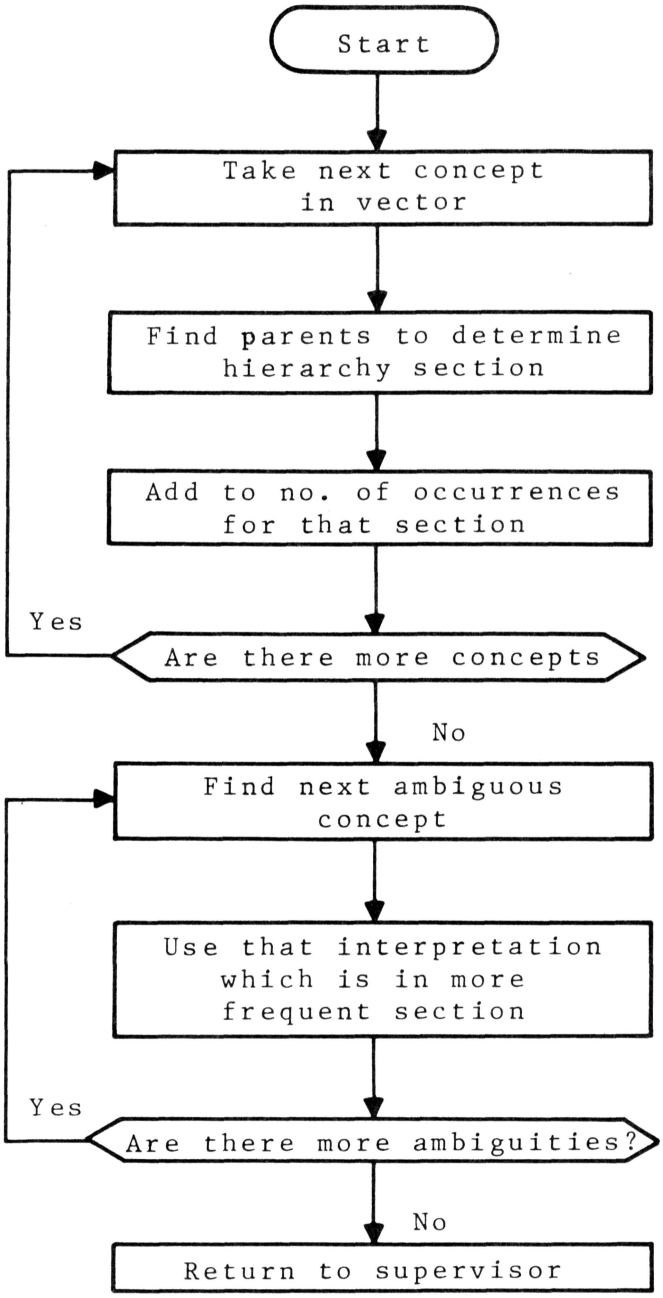
Fig. 6

A more general use of the hierarchy is to resolve ambiguities in the texts. This process is illustrated in Fig. 7. Ambiguous words (those with more than one thesaurus category for the stem concepts) can be resolved by the following rule: when a word is assigned more than one possible thesaurus concept, choose the concept category which belongs to the hierarchy section with the largest number of members in the text. That is, if "IR" is ambiguous as "information-retrieval" or "infra-red", the remainder of the text is examined, and if most of the words are about spectroscopy, "infra-red" is selected. A special table of class frequency is compiled to be used for this algorithm.

F) Concept Vector Formation and Storage

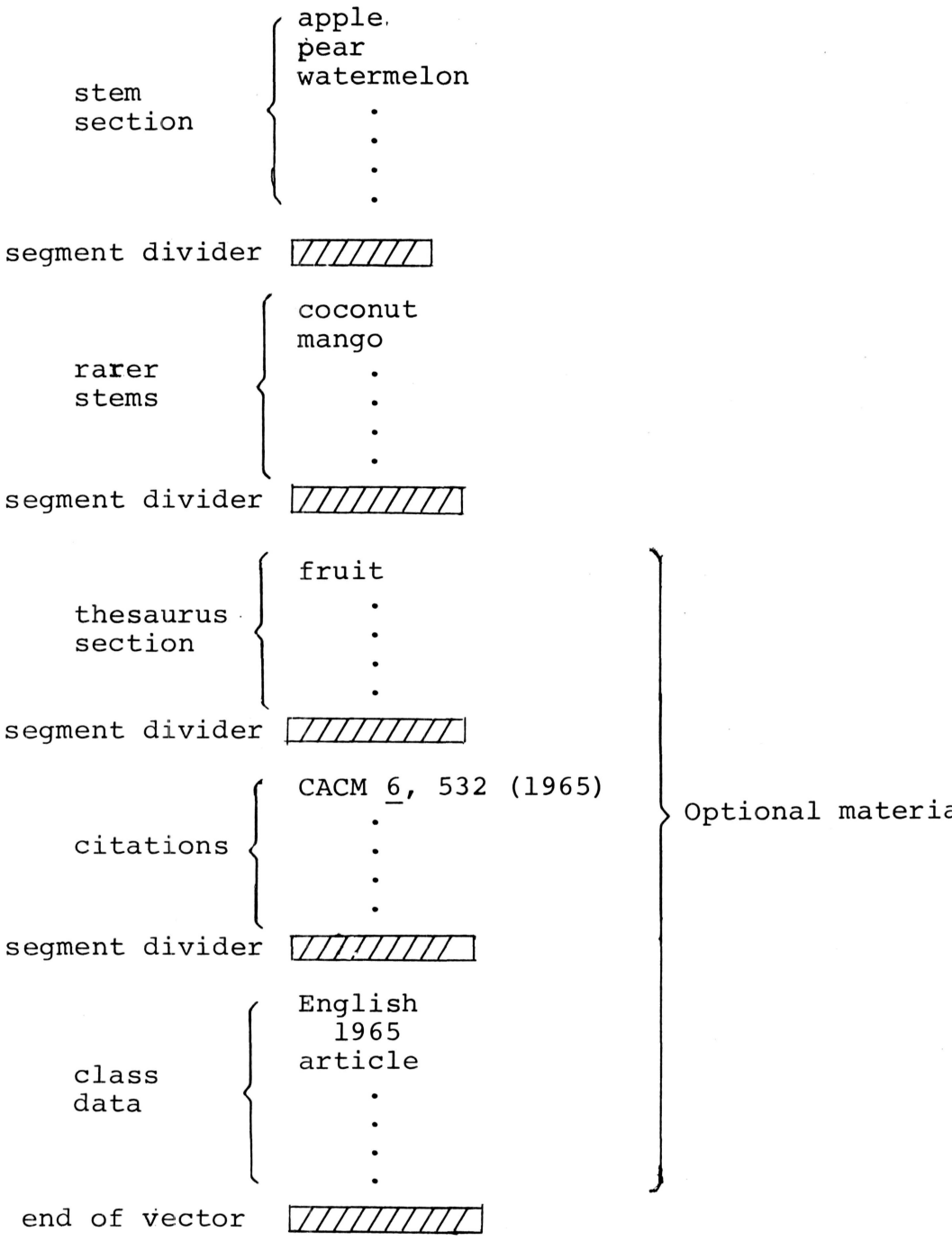
Concept vectors are generated from the concepts produced in the lookup and from any phrase or hierarchy expansions. The concepts are sorted in order, and then condensed to 16 bit halfwords by indicating the higher-order bits trhough a division of the concept vector into segments. That is, a special flag is added to the concept vector, and when this flag is reached, the computer recognizes the beginning of a new concept number sequence. The sequences include: high-frequency stems, low-frequency stems, thesaurus concepts, and other information.

A sample concept vector is shown in Fig. 8. This organization was devised by R. Williamson [4].



Ambiguity Resolution

Fig. 7



Sample Concept Vector

Fig. 8

G) Searching of Document Collections

Previous experience confirms that the cosine correlation is a fast, efficient request-document matching algorithm [5]. The cosine correlation for request r and document d is defined as

$$c_{rd} = \frac{\sum_i r_i \cdot d_i}{\sqrt{\sum r_i^2 \sum d_i^2}}$$

where r_i is the weight of concept i in the request and d_i is the weight of concept i in the document.

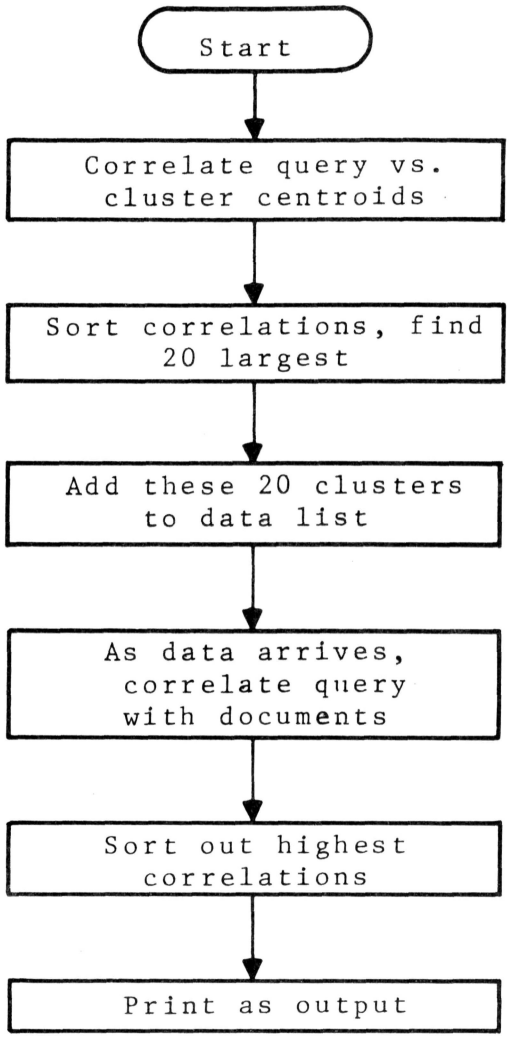
Document vectors should be stored in document order, rather than in an inverted file. The total storage required in each case is similar, but the number of disk accesses for a clustered search is much smaller than the number of disk accesses required for an inverted file. Furthermore, the large sort needed to generate the inverted file represents a long and unattractive use of computer time. In addition, the relevance feedback process requires access to the complete vectors of each document. Furthermore, when the system is expanded to include information such as language, journal, etc., the overhead required to store this information with an inverted file would be very large. It is shown later

in this report that the time required even for a full search on a document file is perfectly reasonable, and thus no good reason seems to exist for the use of an inverted file.

The computation of the correlation coefficient is straightforward. In order to save time in computing the sums of the squares of the weights of each document, this figure is computed at lookup time and added to the document vector.

Facilities for clustered searches are provided in order to save time with large collections. A flowchart is reproduced in Fig. 9.

Each request is first matched against a set of cluster centroids to determine which cluster should be searched. The clusters bearing the closest similarity to the query are then brought in from the disk and searched in full. the total number of documents searched may only include a small fraction of the collection. This saves time and is particularly valuable for such operations as the first search of a collection prior to the use of relevance feedback. The total time required to perform request document correlations with a clustered search depends on the number of clusters generated and used. Consider a document collection of 25,000 documents separated into 100 clusters of 250 documents each



Search With Clusters

Fig. 9

(slightly fewer clusters than the optimum). If 20% of the collection, i.e. the 20 most promising clusters, are to be searched, a request document comparison involves about 100 compares, multiplications and additions. If one allows 10 usec for each operation, request/document correlations may be performed at a rate of 1000 per second. Since 100 cluster centroids and 5000 documents are to be searched, a total of 5100 correlations must be made, representing about 5 seconds of time. The I-ϕ time involved, in the worst case (every cluster plus the centroids requires a separate disk access) is 21 fetches or 2.1 seconds. Most of this work can be overlapped with processing. Note that, in sharp contrast to text lookups, request-document correlations are <u>not</u> I-ϕ limited. The size of the various clusters would be about 250 documents x 50 vector entries x 4 bytes per entry equal to 50,000 bytes or 2-3 blocks of 20,000 bytes. The 100 centroid vectors would require about 20,000 bytes and could conveniently remain in memory during search operations.

Since the total lookup time is of the order of 1/2 second, and the total correlation and search time on the order of 5 seconds, response times of 10 seconds should be feasible for the final system using a clustered search. A full search of the 25,000 document collection would take

about 25 seconds, which is not unreasonable. When collections become larger (250,000 documents) they also become more diverse, so that the total number of documents to be searched may not increase.

H) Clustering of Document Collections

The most time-consuming program of the retrieval system is likely to be the generation of the document clusters. Fortunately, this need be done only at rare intervals. These details of the clustering algorithm are given by Brauen and Messier and are therefore not repeated here [6]. This operation is independent of the remainder of the retrieval system and could be programmed as a separate job.

I) Relevance Feedback

Increases in retrieval effectiveness are obtained with the use of relevance feedback. This procedure is suitable for use by untrained requestors and is ideally adapted to conditions prevailing in an operational system. Previous experiments with feedback have indicated that the best strategy, called "decrement high", involves the addition to the query vector of concepts derived from retrieved relevant documents, and the subtraction from the query of concepts from the highest ranking nonrelevant document [7].

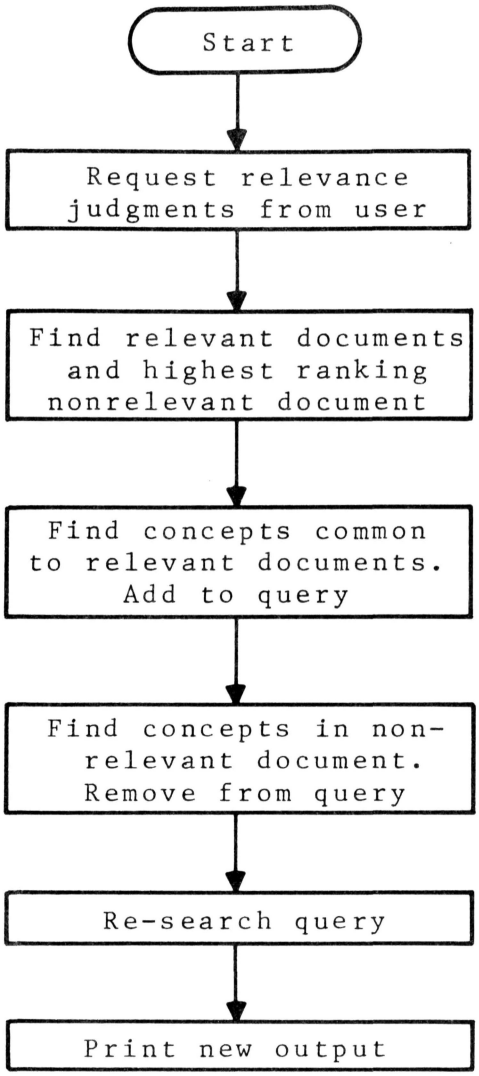
In the actual system, the relevant material used need not be retrieved through a search. Provision is made for the introduction of previously known relevant material, and for the adjustment of the query through feedback without performing a normal retrieval run. This would permit improved performance on the initial retrieval run. In fact, an entire query could be defined using the feedback algorithms together with a set of known relevant documents. The feedback routines could extract the common concepts from the known relevant documents and produce a query that would retrieve them.

A flowchart for the feedback routines is shown in Fig. 10. The additions to the processing and data list are given below

Task List	Data List
ask user for known relevant	console input
retrieve known relevant and adjust query	known relevant document vectors

J) Dictionary Displays

Some users may wish to assist in the optimization of their own queries. The most useful information the system can provide might be a display of the thesaurus and



Feedback Routines

Fig. 10

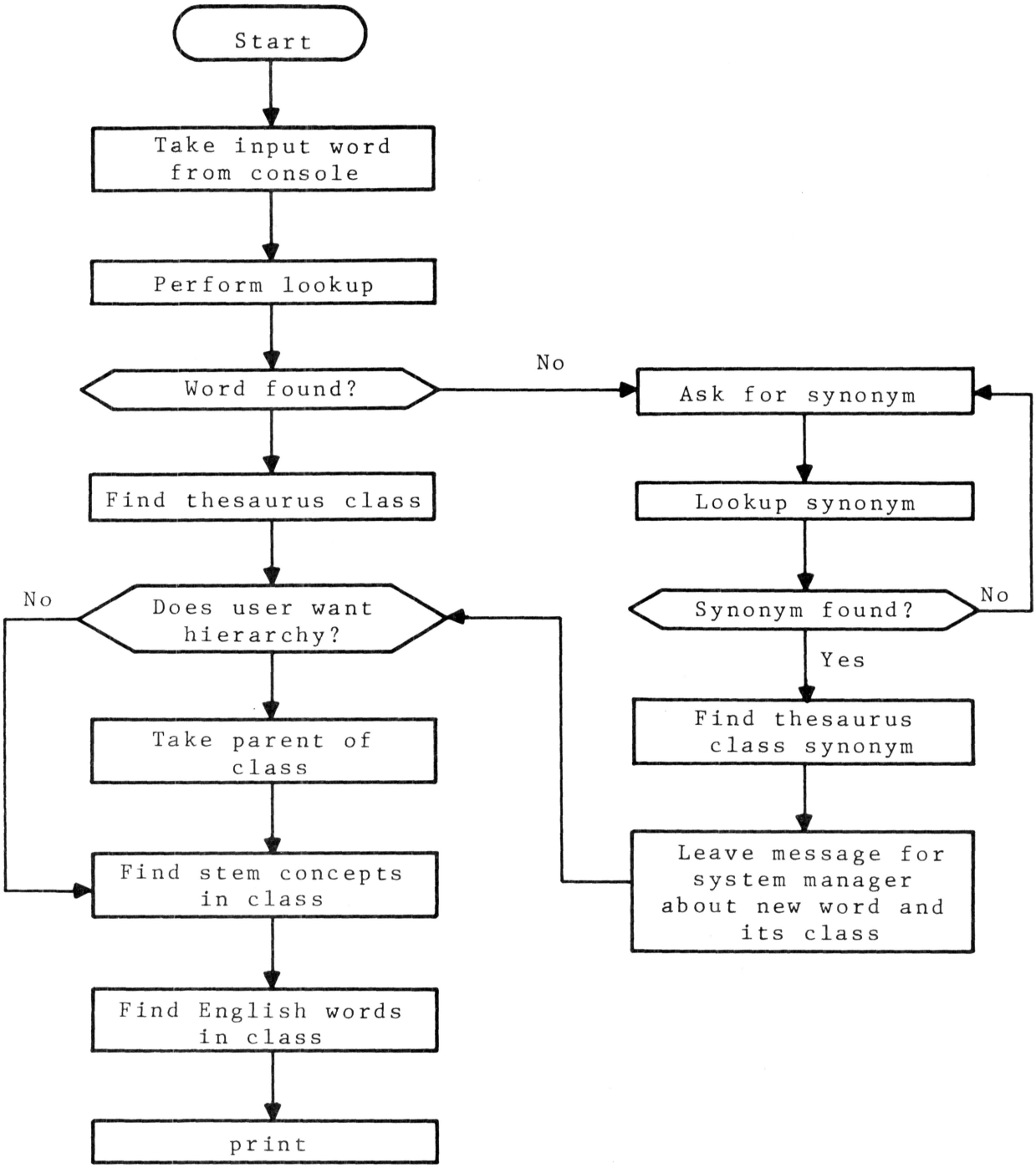
hierarchy categories pertaining to a given subject area [8]. This is done by maintaining a reverse-sorted thesaurus, which is easily generated when the original thesaurus is created. The reverse-sorted thesaurus provides English stems corresponding to the thesaurus categories. The user may type in a word, which can then be looked up in the stem dictionary and thesaurus. The reversed thesaurus is then used to obtain the stem dictionary concepts in that category. The corresponding words can also be printed out. If desired, hierarchical expansion can be performed first to provide access to even more words. In the initial versions of the system, thesaurus display would not be included, saving not only the associated programming but also the need to store a reversed thesaurus and the English version of the stem dictionary. Listings of the thesaurus could instead be provided to interested users. As the system is developed, however, it would probably be desirable to automate the thesaurus. This would be especially desirable if a high-speed, "soft-copy" output device (such as an oscilloscope display) were available. Another useful facility would be a procedure allowing the users to indicate changes that should be made in the thesaurus. Users should not be allowed to change the thesaurus themselves, since it is doubtful that the average user has the ability or the detailed knowledge needed for this rather delicate task,

but a file of "proposed changes", stored on the disk, would be most useful in expanding the thesaurus.

A flowchart of a possible display and update routine is shown in Fig. 11.

K) Citation Searching

The system as described could easily handle citation information, if the citations were available at the input end. It would only be necessary to standardize reference formats, to introduce the citations in some coded form, and to process them through the null dictionary and add them to the concept vectors. For example, the standard four-letter journal codes of Chemical Abstracts could be combined with the volume number and page number to provide a 12-14 character code uniquely identifying literature references. This code could be hashed, but a simpler system would require looking up the journal code in a small table, translating the numbers to binary, and producing a 30-bit code directly. This code could be used as an entry in a separate set of tables, processed in the same way as the hashed English words. The additional "concepts" that would result could then be place in distinct sections of the concept vector and used for retrieval. It would not be necessary for the original query to use any references, since the relevance feedback routines could introduce them from retrieved documents. The value of such information is being studied with the current SMART programs.



Thesaurus Display Routine

Fig. 11

L) Class Information

Articles can also be characterized by various external properties, such as language, date, journals, etc. It is likely that many users would wish to request specific languages that are of interest; or users may wish to reject material published before a certain date; or to specify certain types of articles (e.g. review articles). The final system should, therefore, include facilities for processing this type of information as well. This information is normally used in an absolute sense, and it is doubtful if the correlation coefficients would be of much use. Characterization of the few main items (language, date, type of publication, etc.) seems preferable with simple tests to determine which documents are acceptable.

M) Selective Information Dissemination

No special facilities are needed to add SDI. If those documents which have just been added to the system are identifiable, a standing file of SDI requests could then be processed as a background job once a week with a specification that only new documents should be searched. Alternatively, all new documents could be checked against the standing file as they are received. It might be useful to maintain, for each SDI query, one "nearly-but-not-quite-relevant" document. All new

documents correlating higher than that document would then be retrieved.

N) User Information Files

When the system is eventually completed, and is being used operationally by many users, each user might be allowed to store a special file of information including his peculiarities. It might contain SDI requests which the user could activate from a console; special dictionary transformations for this particular user's area of interest; or specific date and language restrictions. Eventually, the computer might also retain, for each user, records on retrieval methods used and resulting performance. It could then automatically select the correct retrieval algorithms upon recognizing the user's name thereby providing truly personalized service.

4. Equipment

The computer system considered in this section is an IBM 360/65. Both Cornell and Harvard University either have or have ordered this machine, with bulk core. As is demonstrated in part 3, the retrieval system could operate efficiently (with one console only) using 50K bytes of data storage space. Since both Cornell and Harvard have ordered bulk core units, larger storage capacities should be available. The programs, of course, could easily take

up an additional 100K of memory. If the programs do get very large, a possible solution is to divide the programs into blocks and bring them in and out of core as needed. This could be done by the SMART supervisor, or through OS|360, but would be undesirable since it would lead to competition for the disk unit and degrade response time.

The recommended random-access unit is the disk drive. For a collection of 25,000 documents with a dictionary of 50,000 words, one might expect the following total storage needs (8 bit/character):

a) 12,500,000 bytes for English text
b) 400,000 bytes for English dictionary
c) 200,000 bytes for hash dictionary
d) 5,000,000 bytes for looked-up null vectors
e) 4,000,000 bytes for thesaurus vectors.

The total storage requirements are thus about 22 million bytes (with digram enciphermant, about 17 million bytes). However, the English document text need not be kep on disk, lowering the storage requirements to 10 million bytes. There is thus no difficulty in storing the entire system plus data on one disk pack, with a capacity of 29 million bytes [9].

The input-output consoles are initially planned either as teletypes or selectric typewriters. Later improvements might include oscilloscope screens for fast displays and microfilm readers. The microfilm readers could even be computer-driven, allowing the computer to display documents directly. The initial operational console, however, might simply be placed in or near a library.

Should the system be expanded by a factor of 10 or so, the use of a data cell will probably become necessary. The total capacity of a data cell is 400 million bytes [10] which would be adequate to store vectors for 1,000,000 documents.

The only major additional programs needed for system development would be a good text editor and a concordance routine. Both will probably be available from program libraries. Initial coding on the retrieval system should concentrate on the basic routines, lookup, thesaurus, and search. The remaining options can be added as time permits.

5. Summary

The design of an automatic retrieval system for document collections of practical size with both interactive and batch-processing capabilities is shown to be feasible since previous experiments have indicated that fully-automatic systems can provide retrieval performance equivalent to that

of existing systems, [11,12] it is believed that the construction of a completely mechanized documentation center should be initiated.

References

[1] G. Salton and M. E. Lesk, Computer Evaluation of Indexing and Text Processing, Journal of the Association for Computing Machinery, Vol. 15, No. 1, 1968.

[2] D. Murray, A Scatter Storage Scheme for Large Dictionaries, Term Project Report, Computer Science 435, Cornell University, Spring, 1968.

[3] E. M. Keen, Thesaurus,Phrase and Hierarchy Dictionaries, Report ISR-13 to the National Science Foundation, Section VII, Cornell University, Dept. of Computer Science, December 1967.

[4] R. Williamson, private communication.

[5] K. Reitsma and J. Sagalyn, Correlation Measures, Report ISR-13 to the National Science Foundation, Section IV, Dept. of Computer Science, Cornell University, December 1967.

[6] R. T. Grauer and M. Messier, An Evaluation of Rocchio's Clustering Algorithm, Report ISR-12 to the National Science Foundation, Section VI, Dept. of Computer Science, Cornell University, June 1967.

[7] E. Ide, User Interaction With an Automated Information Retrieval System, Report ISR-12 to the National Science Foundation, Section VIII, Dept. of Computer Science, Cornell University, June 1967.

[8] M. E. Lesk and G. Salton, Evaluation of Interactive Search and Retrieval Methods Using Automatic Information Displays, Report ISR-14 to the National Science Foundation, Section IX, Dept. of Computer Science, Cornell University, October 1968.

References
(contd)

[9] IBM System/360 Component Descriptions — 2314 Direct Access Storage Facility, Publication No. A26-3599-2, IBM Corp. 112 East Post Road, White Plains, N. Y. 10601.

[10] IBM System/360 Component Descriptions — 2321 Data Cell Drive, Publication No. A26-5988-3, IBM Corporation, 112 East Post Road, White Plains, N. Y. 10601.

[11] G. Salton, Search and Retrieval Experiments in Real-Time Information Retrieval, Proc. IFIP Congress, 1968, Edinburgh, Scotland, August 1968.

[12] G. Salton and D. K. Williamson, A Comparison Between Manual and Automatic Indexing Methods, Information Storage and Retrieval, Scientific Report No. ISR-14 to the National Science Foundation, Section VI, Cornell University, Dept. of Computer Science, October 1968.

Appendix

Digram Encipherment of English Words

A simple way to economize on the space required to store long lists of English words in a 360 computer with 8-bit bytes (without abandoning the byte structure as would be necessary if 5-bit or 6-bit codes were used) is to store two characters in each byte. This offers nearly twice the efficiency in storage use and permits simple translation to and from the packed format.

To implement this scheme, the 256 possible 8-bit codes are assigned to the 219 most frequent digrams (two-letter pairs) in English, the 26 single letters, the ten numerals, and the hyphen. Although there are 676 possible digrams in English, the list of the 219 most frequent digrams includes all digrams occurring more than o.05% of the time, and covers 97% of all digram occurrences. (This, of course, comes about because several hundred digrams are phonetically forbidden, e.g. VG, QB, LJ). For the IRE collection used with the SMART system, the digram list below would have packed the words with an average density of 4.12 bits per character. Whenever, in encoding, an illegal digram occurs, the individual letter substitutes are used for one letter. For example, the abbreviation AEC contains

the usually rare digram AE, and would be enciphered as "A" + "EC". "Abbreviation", on the other hand, is enciphered normally as "AB" + "BR" + "EV" + "IA" + "TI" + "ON".

It should be noted that the alphabetical order of words enciphered in this way will not be exactly the same as the normal one, but assignment of the digram codes in order, with the single-letter codes near the most similar digrams produces almost-alphabetical ordering (e.g. the single letter code for I should be in the place of the digram II, which is the most frequent digram beginning with I not in the list attached, and between the digram codes for IG and IL, IH, IJ, and IK are even rarer than II).

In theory, even greater economies could be achieved by the use of trigram or higher-order encipherment, but the unit of data would no longer be a byte if any significant number of trigrams were used, and the enciphering and deciphering tables would be much longer. The advantages of introducing a few trigrams or tetragrams (THE, THA, ION, TIO, etc. or TION, MENT, etc.) into an otherwise digram encipherment do not seem to outweigh the additional complexity.

The 225 most frequent English digrams in the IRE text are as follows in decreasing frequency order:

TH	ST	EM	LO	SC	EV	SP	AV	LD
IN	DE	LI	RS	AP	MU	TU	EG	OI
ON	RI	CE	GI	AM	MO	LT	FR	EI
TI	SI	EL	MI	RY	RT	OW	BA	EO
ER	SE	PE	PL	SU	OS	FF	OO	IQ
HE	OM	OL	IG	PP	VA	EP	ZE	RN
TE	ME	AC	OU	RM	EE	FU	NV	GO
AN	LE	PU	UL	YS	RR	MM	TC	FL
RE	TO	HI	IR	OP	UM	SH	CL	XI
AT	NS	LA	QU	BL	UC	DA	SW	YN
OR	UT	AS	CI	PA	IF	LS	TT	CC
ES	MA	BE	IL	SY	AI	NU	GA	NN
CO	NE	NC	UR	AG	VI	IZ	PH	NF
IO	CH	SS	EA	UI	AD	RD	FA	XP
AL	PR	NI	HO	IB	UE	OB	UP	KE
OF	TR	NA	IV	GR	WH	MB	CK	UB
ED	MP	US	IE	VO	OT	WO	UD	RK
IT	TA	OD	FI	XT	LU	WE	RG	SL
EN	DI	TS	PO	AB	MS	TW	BR	RV
RA	FO	HA	OG	UA	GN	BO	EW	RP
IS	CA	UN	WI	OC	DU	DS	DO	DD
AR	CT	IM	LY	RC	EF	PT	WA	DR
NG	EC	EX	SO	CR	BI	SA	GU	CS
NT	VE	GE	IA	EQ	BY	AY	BU	CY
ND	ET	CU	LL	NO	TY	RU	HN	PI

The recommended list of 219 digrams is the above list except for the six least frequent digrams, RP, DD, DR, CS, CY and PI.