A PROGRESS REPORT ON AUTOMATIC INFORMATION RETRIEVAL

by

G. Salton*

TR 79-368

Department of Computer Science
Cornell University
Ithaca, New York 14853

*This study was supported in part by the National Foundation under Grant DSI77-04843.

A Progress Report on Automatic Information Retrieval

G. Salton

Abstract

This study is a state-of-the-art report of work in automatic information retrieval. Various enhancements of operational on-line retrieval systems are described such as the utilization of special front-end devices providing compatibility among different search services, the introduction of fast back-end search devices, and the use of local clustering and query reformulation operations designed to improve retrieval output. Certain new algorithms for fast text matching and for optimum term weighting are also mentioned, as are advances in the construction of theoretical retrieval models.

*Department of Computer Science, Cornell University, Ithaca, New York 14853.

This study was supported in part by the National Foundation under Grant DSI-77-04843.

1. Introduction

The information retrieval area covers several document and data processing operations, including content analysis and indexing, that is, the assignment of keywords and descriptors to records and information items; record classification and file organization, that is the determination of appropriate storage structures for the record collection, and of the needed relationships among the information items; information searching designed to locate individual records within the files; retrieval and dissemination, that is the extraction of certain records from the file for presentation to the user; query negotiation and reformulation used to improve original user query statements in the hope of obtaining additional relevant items or fewer extraneous materials; and finally retrieval evaluation operations assessing the effectiveness of the retrieval output, and the efficiency of the search process.

It may come as a surprise to learn that on the surface at least, little progress is evident in any of these areas. The indexing task is still carried out in operational situations predominantly by human subject experts who attempt to predict in advance the circumstances that may lead the user population to specify particular keywords and descriptors to express various kinds of information needs.

Only two kinds of file organization appear to be in widespread use: a direct, substantially unorganized file in which the documents are stored in chronological order of acquisition, requiring a complete sequential search to identify items to be retrieved; and an inverted file in which an auxiliary (inverted) directory provides for each existing keyword the complete list of record addresses corresponding to that key. In the inverted file system, retrieval is a two-step process consisting of the identification of appropriate record addresses in the directory, followed by retrieval of the corresponding records from the main file.

The query-record comparison is normally based on an exsct match process between the keywords included in the query and the corresponding keywords attached to the information items, the aim being to identify all records exhibiting precisely the keyword combination used in the query statement. Most operational retrieval aystems have provision for some type of query improvement or reformulation; normally this operation utilizes vocabulary displays consisting of terms related to those included in the original query statements. Such related terms may then serve to "expand" query formulations in the hope of retrieving additional, useful output.

Established methods also exist for evaluating search output, often involving some sort of tentative relevance assessment on the part of the user population. When enough "relevant" items are retrieved, and most extraneous items are rejected, the assumption is that the system operations are satisfactory.

Whereas most of the basic information retrieval processes have remained relatively unchanged for about ten years, advances are evident in many secondary areas leading to speedier operations and more effective output products. The main improvements are briefly described in the remainder of this atudy.

2. On-Line Information Retrieval

A discussion of recent developments in information retrieval cannot begin without stressing the obvious connection between information dissemination and on-line searching. In principle, the use of console entry devices for user-system interaction during the search process does not alter the intellectual operations which form the basis of most retrieval activities. In practice, the conversion to on-line operations has greatly increased the commercial viability and the perceived importance of search and retrieval systems as components of the normal research and development activities in science and technology.

There now exist hundreds of different bibliographic data bases available for on-line searching, comprising many tens of millions of records, and many hundreds of search operations are ostensibly carried out every day. [1,2] Furthermore, the extensive coverage now provided in many subject areas together with the relatively low search costs and the absence of substantial waiting times between query introduction and system response create the impression among potentisl users that the search effort is likely to be fruitful.

It is not possible in the present context to outline the detailed characteristics of current operational retrieval systems. Among the techniques that have received general acceptance, the following may be of principal importance:

a) the use of logical connectives (AND,OR,NOT) to interrelate the key terms included in the query statements;

b) the possibility to conduct free text searches where query statements are compared with words occurring in text excerpts, or in full document texts;

c) the use of system displays to guide the user during the query formulations and during other interactions with the system;

d) the possibility to browse through the retrieved output in order to identify useful output items, and to consult various dictionary and vocabulary displays during the search proceas;

d) the possibility to save partially formulated queries and to combine pieces of previously formulated search statements to produce expanded information requests;

f) the use of file creation and file updating facilities to store either vocabulary data such as dictionaries and thesauruses, or bibliographic records consisting of retrieved items, or items subject to special search operations.

Systems that include provisions for free text searching may use a variety of techniques to refine the initial output produced by simple word matching. Thus the location of words within a text -- whether occurring in titles, abstracts, figure captions, and so on -- may be used to assign special importance indicators to certain terms, and these indicators may play a role in the subsequent query-document comparison. Truncated words can be generated by deletion of suffixes and prefixes, and the resulting word stems may then be compared in documents and queries instead of the original words. Word phrases consisting of several individual words can be used for matching purposes instead of single terms, and the corresponding phrase definitions can be based on word location and word distance properties. Word synonyms may be available by consulting preconstructed dictionaries and thesauruses, and hierarchical term expansion may serve to replace the original terms by broader terms, narrower terms, or related terms included in an existing stored term hierarchy. Examples of some of the query formulation facilities included in the well-known Stairs retrievsl system are shown for illustration in Table 1. [3]

Systema that use stored dictionaries may also provide facilities for dictionary generation and updating, and for the specification of appropriate hierarchical relationshipa between term entries. Thus from incomplete user-supplied information, the system may be used automatically to generate complete term groupings as well as hierarchical tree arrangements for governing and dependent terms. For example the Stairs system will automatically recognize that term A is broader than B and C, whenever the user specifies that B and C are narrower than A. [3]

Language processing operations are also important in many special purpose environments. Thus a system designed for the manipulation of bibliographic citations in a library environment may provide special facilities for "rotating" author names and for recognizing different spelling conventions. A typical author name processor is illustrated in the example of Table 2. [4]

The previously mentioned file creating and file manipulation facilities are componenta of special retrieval systems designed to process formatted files of the kind used in business environments. The files are then conceptually represented by tables, and the retrieval output normally depends on individual table entries. For the sample file of Table 3, it would, in principle, be possible to retrieve all author names publishing books in a certain year, or all authors with a given number of publications, or living in a given city. It is not difficult to conceive of retrieval systems in which bibliographic retrieval based on content indicators is combined with a data base management facility using objective table entries for retrieval purposes. [3,5]

3. New Developments

A) Hardware Enchancement of Operational Systems

The relative popularity of on-line retrieval services has generated interest in improving and streamlining system operations. The most obvious possiblities in this connection consist in making it easier for the user to approach the syatem and in eliminating some of the drawbacks inherent in the processing of very large files. The first problem confronting the user is the multiplicity of different commercial retrieval services currently available on the market. In principle, the formats used to represent bibliographic information in machine readable form, and the search and retrieval systems utilizing these data bases,could all be identical in view of the comman intellectual basis of current operations. In practice, many incompatible retrieval services actually exist uaing different command languages and offering somewhat different facilities. This unfriendly situation for the end user implies that search intermediaries must often be interposed between user and system, because only search experta can be expected to become familiar with a variety of incompatible retrieval systems.

Because of the large sums of money invested in the existing systems, it is not likely that voluntary agreement would be forthcoming with respect to common formats and services. Methods can, however, be devised in an attempt to minimize the incompatibilities currently existing in data base formats and retrieval services. Thus, preprocessing routines can be incorporated into existing systems designed to convert the large variety of different input forms to a common processing format. The resulting input can then be fed to a single, standardized retrieval service. [6]

Alternatively, front-end procedures can be designed to translate a standardized input command language into the internal forms used by several different retrieval services. Such a system simplifies the situation for the user since only a single computer interface must now be mastered. Unfortunately, the standardizing front-end represents overhead above and beyond the normal storage and search costs, and the commerical services are not likely to be anxious to absorb the extra cost. A typical front-end interface system known as CONIT is represented in simplified form in Figure 1. [7]

In addition to standardizing the user-system interface, current retrieval operations can in principle also be improved by back-end changes, that is, by providing more efficient methods for query-document comparison and faster keyword searches. Consider the normal situation where search requests are processed as Boolean combinations of keywords (for example, "A or (B and C)"). Such requests can be answered by a sequential scan of all records in the file to identify records with the appropriate keyword combination, or more likely the lists of document references stored in an inverted directory can be treated by list intersection (required for the B and C lists in the earlier example) and list union (needed for the A list and the results of the B and C intersection) in order to identify the relevant document references.

When the keyword lists are long, that is, when many records are identified by a given key, the list intersection and merging operations may be substantially speeded up by building a network of comparator units that can operate in parallel on different elements of the input lists. Thus n comparator units on a given level of the network can simultaneously perform pairwise comparisons between n different input elements from each of two lists. The output from a given level of the network may be fed to additional parallel comparator units on another level of the network. By repeating the parallel operations on all network levels, the list merge operations can be performed from ten to one hundred times faster than would be the case with ordinary processors. [8,9]

When the stored records are compared with the query statement using a sequential scan of the file instead of an inverted file process speed advantages may be obtained in one of two ways. The first consists in partitioning the file into subfiles and providing a separate small processor for each of the subfiles. Each of the auxiliary processors is then charged with performing the query-record comparisons for its particular subfile, and all processes are expected to operate in parallel. In practice, a separate auxiliary processor might be provided for each disk (or each disk face) for a file stored on a disk stack. Appropriate communication channels must, of course, also be provided between the auxiliary parallel processors and the central processing unit which handles the user programs. A typical arrangement for parallel back-end processors is shown in Figure 2. [10,11]

The second possiblity for producing efficiency improvements in scanning a file of records consists in using an associative memory device. An associative memory of n cells can store n different keywords, and all these keywords are then matched in parallel with the contents of a sequentially scanned main file. Typically, the keywords included in a given search request are stored in the cells of the associative memory, and all these keywords are then compared against the record contents in a single scan of the main file. A typical arrangement of this kind is shown in Figure 3. [12,13,14]

Auxiliary search processors and associative memories designed to speed up the query-record comparison operations are attractive in many retrieval environments, because the query-record matching may be expected to consume a large part of the available resources. Retrieval situations do, however, exist where global matches between complete keyword sets (vectors) attached to documents and queries are used to measure query-record affinities. More specifically, the global matching method replaces the normal exact match between specific keyworda by the computation of a variable similarity coefficient that reflects the degree of sgreement between the corresponding keyword sets; items may then be retrieved when the similarity coefficients with the query vectors are sufficiently high. [15,16]

The vector processing model based on similarity computations requires substantial computational ability, because a variety of distance-finding and pattern matching operations must then be carried out. In such circumstances, the enchanced computational facilities provided by one of the modern array processors may be especially helpful. Array processors are special arithmetic processing units often comprising adders, multipliers, as well as dedicated registers and storage units. In general a number of computations may be overlapped or carried out in parallel. [17] For example, if a query Q consists of individual terms $q_1, q_2, \ldots q_n$, and a document D is similarly represented by terms $d_1, d_2, \ldots d_n$, the inner product $\sum_{i=1}^{n} q_i d_i$ may be used to reflect the similarity between Q and D. When an array processor is available, several of the multiplications and additions needed for the inner product computation may be carried out simultaneously.

The use of array processor remains to be justified in retrieval. In principle, large efficiency improvements should result.

B) Software Enhancements of Operational Systems

A comparison between some of the operational retrieval systems currently in widespread use, and certain advanced laboratory-type systems indicstes that improvements in operational practice are possible on many fronts. This includes specifically the use of more modern indexing techniques and more advanced file organizations. From a practical point of view, however, the most immediately productive possibility for obtaining gains in retrieval effectiveness may consist in providing sophisticated query negotiation and query reformulation methods.

In operational retrieval systems, the query reformulation system is normally based on vocabulary displays consisting of lists of synonymous terms and terms related to those included in an original query statement. Since the search results are immediately available to the user population in an on-line environment, a query reformulation system can utilize information about previously retrieved documents. In general, one would like to rephrase the query in such a way that the final statement appears similar to documents previously retrieved and assumed to be relevant, while at the same time the query formulation differs from retrieved items known to be extraneous. The well-known relevance feedback method used in the Smart retrieval environment performs that query alteration automatically by increasing the weight of query terms included in the relevant items and simultaneously decreasing the importance of terms found in the nonrelevant items. [18,19] The relevance feedback operstion is symbolically represented in Figure 4 where distances between items on the page are assumed inversely related to the similarity in subject matter, and hence to the similarity in the respective keyword formulations.

The relevance feedback operation can easily be adapted to standard retrieval environments. A system used by the documentation service of the European Community, operates approximately as follows: [20]

a) the term sets attached to previously retrieved items identified as relevant are examined; for each relevant document a query statement is constructed consisting of the "anded" terms (for example [A and B and C and D]);

b) for each such term set, a broadened, partial query is constructed by removing one of more terms (for example, [A and B and C]);

c) these partial statements are used as new queries and additional documents are retrieved;

d) the relevance of the newly retrieved items is assessed and for each partial query a query quality factor is computed defined as the number of new relevant retrieved divided by the number of new nonrelevant items retrieved;

e) for partial queries with sufficiently high query quality factors steps a), b) and c) are repeated.

It is not hard to see that many variations are possible for such a reformulation system. The method actually outlined apparently produces important improvements in retrieval effectiveness. [20,21]

In addition to the query reformulation processes mentioned earlier, a further possibility consists in using local term clustering operations for feedback purposes. It is well known that groups or clusters of terms can be generated automatically. The resulting "thesaurus" exhibits classes composed of terms with similar document assignments. Analogously one can automatically produce document classifications by grouping into common classes all records with similar, or related term assignments. [22,23] From a practical viewpoint automatic classification methods -- whether applied to terms or to documents -- may be difficult to justify because of the expense involved in carrying out these processes on a large scale. [24] These objections do not however apply to local clustering, where the procedure may be applied to a small subfile consisting preferably of documents retrieved in a previous search operation or of terms included in such documents.

A typical query reformulation process based on local term clustering might operate as follows: [25]

a) relevance assessments are obtained for certain documents retrieved in earlier search operations;

b) terms from the documenta identified as relevant are ranked in decreasing order of relative frequency (frequency of occurrence in the documents divided by total frequency in the collection), and terms with large relative frequency are used for query reformulation purposes;

c) alternatively, or in addition, term similarity coefficients may be obtained for pairs of terms occurring in the relevant documents, the size of the coefficients being dependent on common occurrence patterns in the respective documents or document sentences;

d) terms with large enough similarity coefficents are then clustered, and each original query term is considered as the kernel of a cluster of related terms to be used for query reformulation purposes.

Once again many different procedures can be applied locally to previously retrieved documents, or terms, in an effort to improve the presentation of the output and the effectiveness of the retrieval operations. [26] Local operations can be used in particular to rank the retrieved documents in decreasing order of presumed relevance. In one experimental retrieval system, the usual unranked output is first obtained based on exact matching of document keywords with Boolean search requests. A ranked output is then generated by computing for each document $D = (d_1, d_2, \ldots, d_n)$ the cosine similarity coefficient with the corresponding query $Q = (q_1, q_2, \ldots q_n)$, defined as

$\sum_{i=1}^{n} d_i q_i \Big/ \sqrt{\sum_{i=1}^{n} d_i^2 \cdot \sum_{i=1}^{n} q_i^2}$. This renders possible the presentation of the output in decreasing order of the cosine coefficient, that is in decreasing order of the global query-document similarity. [27]

The method used to superimpose the cosine computation on top of the normal inverted file processing consists in storing in each inverted list of document references the weights of the corresponding term in each particular document. A typical inverted list for a specific term T then consists of pairs of values ($D_i w_i$; $D_j w_j$; $D_k w_k$; etc.) indicating that term T has value w_i in document D_i, value w_j in item D_j, and so on. The computation of the numerator of the cosine measure $\Sigma\, d_i q_i$ then becomes possible following consultation of the inverted lists of document references. A separate "document length" file is used to store the length $\Sigma\, d_i^{\,2}$ for each document D. Since the query length $\Sigma\, q_i^{\,2}$ is available directly, the cosine measure can be computed without problems. Measures other than the cosine could be used equally for document ranking and query reformulation purposes.

4. Theoretical Advances

A) Fast Algorithms

In recent years the study of computer algorithms has become increasingly popular among computer experts. In many cases these studies produce discouraging results in the sense that methodologies of great practical importance turn out to be computationally difficult and time consuming. Exceptions do, however, exist and some algorithms of immediate applicability in information retrieval appear surprisingly simple and fast. One such is the string matching process which can be used to match input patterns (that is, query keywords) against strings of text words. String matching is used in many full-text search systems where texts, or text sentences, containing specific keywords, or keyword combinations, must be identified.

The standard, somewhat obvious string matching method begins by comparing the first characters of input pattern and text string, next the second characters assuming that the first characters match, and so on down, until either a character mismatch occurs, or the full pattern is correctly matched by string characters. When a mismatch occurs between a pattern and string character pair, one must backtrack, shift the pattern over by one place, and recommence by comparing this time the initial pattern character with the second string character, and so on for subsequent character pairs. It is not hard to see that this process can become inordinately expensive because of the backtracking and restart operations. Thus, consider an example consisting of a pattern p starting with n a's followed by a b, and a string s beginning with 2n a's followed by a b, as follows

$$\text{pattern } p: \ \overbrace{a\ a\ldots\ldots a}^{n}\ b$$
$$\text{string } s: \ \overbrace{a\ a\ldots\ldots\ldots a}^{2n}\ b.$$

For such a somewhat pathological case, $(n+1)^2$ character comparisons are needed before a full match is finally obtained.

Instead of using the previously illustrated brute-force method, algorithms have been devised which operate in linear or even in sublinear time, that is, the number of character comparisons needed does not exceed the length of the string (instead of the square of the length as in the earlier example), or may even be substantially smaller than the string length. [28,29,30] These methods are normally based on an analysis of the input pattern (that is, of the query keywords) prior to the search operation, resulting in the construction of auxiliary tables indicating the occurrence positions of each individual character in the search patterns, as well as all possible character subsets within the patterns.

Consider, for example, the query pattern "AT-THAT" and let the character positions be numbered from 1 to 7 as shown:

1 2 3 4 5 6 7

A T - T H A T

The auxiliary tables would then show that the character T occurs in positions 2, 4, 7; A occurs in positions 1 and 6; AT occurs in positions 12 and 67; HAT occurs in position 567 only, and so. Whenever a mismatch occurs between two particular characters in the course of the matching . ocess, the auxiliary position information is used to determine the next character pair that needs to be subjected to a comparison operation. Normally the distance between the current string character and the one next to be compared is much larger than a single character position.

Consider the Boyer and Moore method as an example of a very fast string matching process. [30] The algorithm proceeds by first comparing the right-most (rather than the left-most) character in the pattern with a particular character in the string. If a match is found, a left shift is made and the character to the left of the right-most pattern character is treated. A string pointer is used to designate the string character that must currently be matched with the given pattern character. This pointer is moved left by one character position when a match occurs; when a character mismatch occurs, the string pointer is moved to the right so as to designate the next string character to be compared with the right-most pattern character. The principal complexity in the Boyer and Moore algorithm consists in determining exactly the pointer shift (in number of string characters to be skipped) in the event of mismatch.

Consider a string matching example taken from the original reference [30]:

```
pattern p :  AT-THAT
string  s :  WHICH-FINALLY-HALTS.--AT-THAT-POINT...
                   ↑
```

The initial string pointer position (designated by ↑) identifies the nth string character for a pattern of length n. In the example, an attempt is made to match the seventh string character (F) with pattern character T. A mismatch occurs in the example. A teat is now made to determine whether the nonmatching string character (F) occurs anywhere else in the pattern; since the answer is negative, the current string character will necessarily produce mismatches with all other parts of the pattern. Hence, the pointer can be shifted to the right by the full length of the pattern (a shift of seven positions for the example at hand). Such a shift will then prevent a comparison operation between the current string character and any other part of the pattern.

The following situation results after the first pointer shift of n positions:

```
pattern p :         AT-THAT
string  s :  WHICH-FINALLY-HALTS.--AT-THAT-POINT...
                          ↑
```

An attempt is thus made to match the fourteenth string character (-) with the right-most pattern character (T). Once again a mismatch is produced; however, the current string character (a hyphen) does occur in the pattern. Hence the pointer should be ahifted to the right in such a way that the current atring character (-) is aligned with the rightmost occurrence of (-) in the pattern. For the example at hand, a pointer shift of 4 places to the right will align the two hyphens. A right shift aligning a nonmatching string character with the next occurrence of that character in the pattern is known as a Δ_1 shift. The following situation result for the example (note the aligned hyphens):

```
pattern p :             AT-THAT
string  s :  WHICH-FINALLY-HALTS.--AT-THAT-POINT...
                          ✓   ↑
```

The attempt is now made once again to match the current string character (T) with the right most pattern character (T). Since the characters match, the pointer is shifted to the left by one place and the next string character L is compared with pattern character A.

pattern p : AT-THAT

string s " WHICH-FINALLY-HALTS.--AT-THAT-POINT...

L does not match A. At this point two different rules may be invoked:

i) on the one hand, an attempt can be made to find a match in the pattern for the currently mismatched string character; this can be accomplished by using the Δ_1 pointer shift described earlier;

ii) on the other hand, an attempt can be made to duplicate the character matches already determined between certain string characters and a portion of the pattern (T in the example); this is done by locating a new occurrence of the matching string characters further to the left in the pattern, followed by a position shift aligning the previously matching character portions (the Δ_2 pointer shift).

For the example at hand, the nonmatching string character (L) occurs nowhere in the patterns; hence a complete right shift of seven places can be used according to rule (i). According to rule (ii), the right shift would align the matching portions (T); this requires a shift of 3 places (from the rightmost occurrence of T in the pattern to the next occurrence of that same T). Since the pointer had earlier been moved left by an amount corresponding to the already matching subpattern (1 place in the example), that shift must be added to define a total Δ_2 pointer shift of 3 plus 1, that is 4 places.

The two conditions (i) and (ii) are independent of each other and must both be satisfied if the complete pattern is eventually to match a given string portion. Hence, when both Δ_1 and Δ_2 shifts are possible, it is safe to use the maximum between Δ_1 and Δ_2, thereby saving compare time. For the example at hand, Δ_1 is equal to 7, while Δ_2 is only 4. A shift of 7 is thus made producing the following situation:

```
pattern p :                    AT-THAT
string  s :  WHICH-FINALLY-HALTS.--AT-THAT-POINT...
                                        ↑
```

Since the T characters match, a left shift of the pointer next produces a match for the A pair; a further left shift produces a mismatch between string character - and pattern character H. The following situation now prevails:

```
pattern p :                    AT-THAT
string  s :  WHICH-FINALLY-HALTS.--AT-THAT-POINT...
                                      ↑✓✓
```

At this point, the Δ_1 shift needed to match the mismatched hyphen with an occurrence of the hyphen in the pattern is equal to 2 plus 2 extra shifts to the right to compensate for the two left moves occasioned by the previously matching subpattern. Thus Δ_1 produces a shift of 4 and the following situation:

```
p :     AT-THAT
s :  HALTS.--AT-THAT-POINT....
            ✓  ↑
```

The alternative Δ_2 shift would replicate the match for the subpattern (AT) that was found to match earlier. This requires a shift of 5 because the subpattern AT recurs in the pattern 5 places to the left; an extra pointer shift of 2 must be added to compensate for the left moves occasioned by the previously matching subpattern. Thus the total Δ_2 shift equals 7. The following situation results:

```
pattern p :                      AT-THIS
string s : WHICH-FINALLY-HALTS.--AT-THIS-POINT...
                                 √√    ↑
```

It is clear that this latter situation produces a correct match between pattern and string.

To summarize, the string matching process is based on two independent circumstances:

a) a Δ_1 shift is used to match a currently mismatched string character with the next leftward occurrence of that character in the pattern; as a special case, a full Δ_1 shift equal to the complete pattern length is carried out when the mismatched string character does not occur in the pattern;

b) a Δ_2 shift is used to match a previously matching subportion with another occurrence of the same characters in the pattern.

The Δ_1 shift thus depends on a given mismatched string character, whereas the Δ_2 shift depends on the substring reoccurrences within the pattern. In each case, the actual shift used is the maximum between Δ_1 and Δ_2.

Normally, the number of character mismatches is much larger than the number of character matches. In such a case, the number of character comparisons will be substantially less than the string length. For the example used earlier, the string positions actually used for character comparisons (prior to the final successful match) are designated by pointer markers as follows:

```
string s: WHICH-FINALLY-HALTS.--AT-THIS-POINT...
                ↑      ↑  ↑↑   ↑↑↑
```

B) Theoretical Retrieval Models

A great deal of work has been devoted in recent years to the construction of abstract models of retrieval processes. Various approaches have been used including vector theoretical models, fuzzy set theory, and models based on probabilistic and decision-theoretic considerations. [31] A brief discussion showing some of the possibilities inherent in the decision theoretic approach must suffice for present purposes. [32,33]

Consider a vector $X = (X_1, X_2, \ldots, X_n)$ where the X_i are assumed to be random variables. Let $P(X)$ and $R(X)$ represent the probability that a relevant, and a nonrelevant item, respectively, has vector representation X. That is,

$$P(X) = \Pr\{X | X \text{ relevant}\}.$$

Let a_1 and a_2 represent loss parameters associated with the retrieval of a nonrelevant, and the nonretrieval of a relevant item, respectively. Then the average probability of error is minimized by retrieving item X whenever

$$P(X) \cdot a_2 \geq R(X) \cdot a_1 \qquad (1)$$

that is whenever the loss incurred by nonretrieval of a potentially relevant item exceeds the loss caused by retrieval of a potentially nonrelevant one. The decision-theoretic retrieval rule thus specifies retrieval when $P(X)/R(X) \geq a_1/a_2$.

This rule implies also that the preferred order of retrieval is in decreasing $P(X)/R(X)$ order, because when $P(X)/R(X) > P(Y)/R(Y)$ for two vectors X and Y, it is clear that X exhibits a higher relevance probability than Y.

The foregoing retrieval rule is of course useless unless the probability distributions $P(X)$ and $R(X)$ are known for the vectors at hand. Varioua special cases have been studied, the best known being the <u>binary independent model</u>. This model makes the following assumptions:

a) a record is assumed to be an n-dimensional binary vector of terms (all vector elements being equal to either 0 or 1);

b) the terms are assumed to occur independently of each other in the relevant records;

c) the terms are assumed to occur independently of each other in the nonrelevant records.

Assuming term independence and binary vectors, the conditional relevance and nonrelevance probabilities P(X) and R(X) can now be expressed aa

$$P(X) = \prod_{i=1}^{n} p_i^{X_i} (1-p_i)^{1-X_i} \qquad X_i = 0,1$$

and

$$R(X) = \prod_{i=1}^{n} q_i^{X_i} (1-q_i)^{1-X_i} \qquad (2)$$

where p_i and q_i represent the probabilities that an item in the relevant class, and in the nonrelevant class respectively contains term i. Taking logarithms and substituting the expression (2) for P(X) and R(X) in (1) above, the decision-theoretic retrieval function for the binary independent case can be written as follows:

$$\log P(X)/R(X) = \sum_{i=1}^{n} \log[(p_i/1-p_i) \div (q_i/(1-q_i)] \; X_i + \text{constants.} \quad (3)$$

Expression (3) is in the form of the standard inner product similarity function which determines the similarity between query vector Q and document vector D as $f(Q,D) = \sum_{i=1}^{n} w_i x_i$, where w_i the weight of the ith query term becomes $\log\{(p_i/1-p_i) \div (q_i/1-q_i)\}$.

This development shows that the best query statement for the binary independent case haa query weights corresponding to the ratio of the proportion of relevant items containing term i to the proportion of nonrelevant containing term i. This function, known as term precision, or term relevance, has been studied and a number of formal results are available proving the effectiveness of this query weighting system. [34,35]

The retrieval function of expression (3) and the corresponding query weighting system cannot of course be used unless the p_i and q_i values are known. To estimate the p_i and q_i values one can assume that the terms are assigned to. the document collection according to the well-known normal or Poisson distributions; alternatively, in an interactive retrieval situation, the usual relevance feedback process supplies approximations to the p_i and q_i values for many terms contained in previously retrieved documents.

Consider as an example the occurrence characteristics for term i across a set of relevant and nonrelevant documents shown in Table 4. It may be seen that term i is present in r_i relevant documents out of a total of R-relevant; term i also appears in $n_i - r_i$ nonrelevant items out of N-R nonrelevant ones. The optimal query term weighting system now produces the following values for term i:

$$w_i = \log\{(p_i/1-p_i) \div (q_i/1-q_i)\}$$
$$= \log\{(r_i/R-r_i) \div (n_i-r_i/N-R-n_i+r_i)\}$$

When the term assignment to the documents is not independent in the relevant and nonrelevant document classes, the expressions for P(X) and R(X) become complicated. Approximations are normally used which take into account only some of the term dependencies. For example, one can assume that

$$P(X) = P(X_1,\ldots,X_n) = P(X_1)P(X_2|X_1)P(X_3|X_1)$$
$$\ldots.P(X_n|X_{n-1},\ldots,X_1)$$

or

$$P(X) = \prod_{i=1}^{n} p_i^{X_i}(1-p_i)^{1-X_i}\ [1+A]$$

where A is an expression which reflects some of term dependencies. [36,37]

For the binary independent case, the foregoing development can be used to analyze clustered search systems, indexing strategies, and relevance feedback operations. The following expressions are important in this connection and point the way to a more complete development. [33]

Let X_k represent a random variable defined on the kth query attribute, where

$$P_r\{X_k=1\} = s_k$$
$$P_r\{X_k=0\} = 1-s_k \ .$$

If the similarity function $f(Q,D)$ between a binary document and a binary query containing the first ℓ attributes is the random variable $Z = \sum_{i=1}^{\ell} Z_i$ (that is, the sum of the matching terms), the probability that Z, or $f(Q,D)$ is equal to i, assuming term independence will be

$$\sum_{\binom{\ell}{i}} \left(\prod_{j=1}^{i} s_{k_j} \right) \left(\prod_{j=i+1}^{\ell} (1-s_{k_j}) \right) = C(S_1,\ldots,S_\ell,i).$$

Assuming once again two classes of R relevant and N-R nonrelevant items, and using the probability parameters p_k and q_k defined earlier, the following expressions are readily derived for retrieval cut-off i, that is, for a system in which items with at least i matching query terms are retrieved:

a) the expected number of relevant items retrieved

$$E_1 = R \sum_{j=i}^{\ell} C(p_1,\ldots,p_\ell,j)$$

b) the expected number of nonrelevant retrieved

$$E_2 = (N-R) \sum_{j=i}^{\ell} C(q_1,\ldots,q_\ell,j)$$

c) the recall, that is, the probability that an item is retrieved given that it is relevant

$$Re = \sum_{j=i}^{\ell} C(p_1,\ldots,p_\ell,j) = r_i/R$$

d) the fallout, that is, the probability that an item is retrieved given that it is nonrelevant

$$Fa = \sum_{j=i}^{\ell} C(q_1,\ldots,q_\ell,j) = n_i-r_i/N-R$$

e) the precision, that is, the probability that an item is relevant given that it is retrieved

$$Pn = E_1/E_1+E_2 = r_i/n_i \ .$$

It is not difficult to use these expressions to gain a good understanding of the retrieval process for the binary independent case. [38,39,40]
For other retrieval situations, more complicated models must be used, and much of the analysis still remains to be carried out.

One may hope in conclusion that realistic abstract retrieval models may eventually emerge to help in understanding the retrieval process more completely. In the meantime, further gains in retrieval effectiveness will no doubt be obtained by utilizing practical tools including in particular the query reformulation and local clustering operations previously described, and by considering the new possibilities in retrieval software and hardware.

References

[1] M. E. Williams, Networks for On-Line Data Base Access, Journal of the ASIS, Vol. 28, No. 5, September 1977, p. 247-253.

[2] M. E. Williams, The Impact of Machine-Readable Data Bases on Library and Information Services, Information Processing and Management, Vol. 13, No. 2, 1977, p. 95-107.

[3] IBM Scientific and Cross Industry Center, Stairs VS-A Tool for the End User, IBM Nederland, Uithoorn, June 1978.

[4] Project Ballots, Stanford University's Ballots System, Journal of Library Automation, Vol. 8, No. 1, March 1975, p. 31-50.

[5] R. T. Dattola, First - Flexible Information Retrieval Facility for Text, Journal of the ASIS, Vol. 30, No. 1, January 1979, p. 9-14.

[6] M. E. Williams, Experiences of IIT Research Institute in Operating a Computerized Retrieval System for Searching a Variety of Data Bases, Information Storage and Retrieval, Vol. 8, No. 2, April 1972, p. 57-75.

[7] R. S. Marcus and J. F. Reintjes, The Networking of Interactive Bibliographic Retrieval Systems, Report ESL-R-656, Electronic Systems Laboratory, MIT, Cambridge, Mass., March 1976.

[8] L. A. Hollaar, Specialized Merge Processor Networks for Combining Sorted Lists, ACM Transactions on Database Systems, Vol. 3, No. 3, September 1978, p. 272-284.

[9] L. A. Hollaar and W. H. Stellhorn, A Specialized Architecture for Textual Information Retrieval, Proc. National Computer Conference, 1977, p. 697-702.

[10] E. Ozkarahan, S. Schuster, and K. Smith, A Data Base Processor, Report CRSG-43, Computer Systems Research Group, University of Toronto, September 1974.

[11] C. Lin, D. Smith, and J. Smith, The Design of a Rotating Associative Memory for Relational Database Applications, ACM Transactions on Data Base Systems, March 1976, Vol. 1, No. 1, p. 53.

[12] E. S. Davis, STARAN Parallel Processor System Software, Proceedings AFIPS 1974, Vol. 43, AFIPS Press, Montvale, NJ, p. 16-22.

[13] J. A. Rudolph, A Production Implementation of an Associative Array Processor-STARAN, Proc. AFIPS 1972, Vol. 41, Part 1, AFIPS Press, Montvale, NJ, p. 229-241.

[14] J. Minker, Associative Memories and Processors -- A Description and Appraisal, Report TR-195, Computer Science Dept., University of Maryland, July 1972.

[15] G. Salton, editor, The Smart System -- Experiments in Automatic Document Processing, Prentice Hall, 1971.

[16] G. Salton, Dynamic Information and Library Processing, Prentice Hall, 1975.

[17] A. L. Robinson, Array Processors: Maxi Number Crunching for a Mini Price, Science, Vol. 203, No. 4376, 12 January 1979, p. 156-160.

[18] J. J. Rocchio, Jr., Relevance Feedback in Information Retrieval, in The Smart System-Experiments in Automatic Document Processing, G. Salton, editor, Prentice Hall Inc., Englewood Cliffs, NJ, 1971, Chapter 14.

[19] E. Ide and G. Salton, Interactive Search Strategies and Dynamic File Organization, in The Smart System, G. Salton, editor, Prentice Hall, Englewood Cliffa, NJ, 1971, Chapter 16.

[20] C. Vernimb, Automatic Query Adjustment in Document Retrieval, Information Processing and Management, Vol. 13, No. 6, 1977, p. 339-353.

[21] R. N. Oddy, Information Retrieval through Man-Machine Dialogue, Journal of Documentation, Vol. 33, No. 1, March 1977, p. 1-14.

[22] G. Salton, Generation and Search of Clustered Files, ACM Transactions on Data Base Systems, Vol. 3, No. 4, December 1978, p. 321-346.

[23] K. Sparck Jonea, Automatic Keyword Classification, Butterworths, London, 1971.

[24] W. C. Croft, Clustering Large Files of Documents Using the Single Link Method, Journal of the ASIS, Vol. 28, No. 6, November 1977, p. 341-344.

[25] L. Attar and A. S. Fraenkel, Local Feedback in Full-Text Retrieval Systems, Journal of the ACM, Vol. 24, No. 3, July 1977, p. 397-417.

[26] T. E. Doszkocs, AID - An Associative Interactive Dictionary for On-Line Searching, On-Line Review, Vol. 2, No. 2, 1978, p. 163-173.

[27] T. Noreault, M. Koll, and M. McGill, Automatic Ranked Output from Boolean Searches in SIRE, Journal of the ASIS, Vol. 28, No. 6, November 1977, p. 333-339.

[28] D. E. Knuth, J. H. Morris, and V. R. Pratt, Fast Pattern Matching in Strings, SIAM Journal of Computing, Vol. 6, No. 2, June 1977, p. 323-350.

[29] A. V. Aho and M. J. Corasick, Efficient String Matching: An Aid to Bibliographic Search, Communications of the ACM, Vol. 18, No. 6, June 1975, p. 333-340.

[30] R. S. Boyer and J. S. Moore, A Fast String Searching Algorithm, Communications of the ACM, Vol. 20, No. 10, October 1977, p. 762-772.

[31] G. Salton, Mathematics and Information Retrieval, to be published in Journal of Documentation.

[32] D. H. Kraft and A. Bookstein, Evaluation of Information Retrieval Systems A Decision Theory Approach, Journal of the ASIS, Vol. 29, No. 1, January 1978, p. 31-40.

[33] C. T. Yu, W. S. Luk and M. K. Siu, On Models of Information Retrieval Processes, to be published in Information Systems.

[34] C. T. Yu and G. Salton, Precision Weighting - An Effective Automatic Indexing Method, Journal of the ACM, Vol. 23, No. 1, January 1976, p. 76-

[35] S. E. Robertson and K. Sparck Jones, Relevance Weighting of Search Terms, Journal of the ASIS, Vol. 23, No. 1, May-June 1976, p. 129-146.

[36] C. J. Van Rijsbergen, A Theoretical Basis for the Use of Cooccurrence Data in Information Retrieval, Journal of Documentation, Vol. 33, No. 2, June 1977, p. 106-119.

[37] D. J. Harper and C. J. Van Rijsbergen, An Evaluation of Feedback in Document Retrieval Using Cooccurrence Data, Journal of Documentation, Vol. 34, No. 3, September 1978, p. 189-216.

[38] C. T. Yu, G. Salton, and M. K. Siu, Effective Automatic indexing Using Term Addition and Deletion, Journal of the ACM, Vol. 25, No. 2, April 1978, p. 210-225.

[39] C. T. Yu, W. S. Luk, and T. Y. Cheung, A Statistical Model for Relevance Feedback in Information Retrieval, Journal of the ACM, Vol. 23, No. 2, April 1976, p. 273-286.

[40] C. T. Yu, W. S. Luk, and M. K. Siu, On the Estimation of the Number of Desired Records with Respect to a Given Query, ACM Transactions on Data Base Systems, Vol. 3, No. 1, March 1978, p. 41-56.

	Typical Query Statement	Explanation
1	HEALTH RESORT <u>and</u> RHEUMATIC	Use of boolean connectives
2	1 <u>and</u> FISHING <u>and</u> NORTHERN <u>adjacent</u> BLACK <u>adjacent</u> FOREST	Refines query statement 1 ; adjacency requirements are specified for text worda
3	2 <u>and</u> HOTEL <u>within sentence</u> QUIET	Refines query statement 2; requires terms to occur in same sentence
4	SELECT 3 <u>with</u> ALTITUDE <u>greater than</u> 300	Uses magnitude relations for numeric values of certain keywords

keywords are capitalized
underlined words are reserved by system

Typical Text Search Statements in STAIRS

(adapted from [3])

TABLE 1

Potentially Equivalent Author Names	Explanation
WHITE, J.E.M.	-----
WHITE, J E M	punctuation deleted
WHITE, M	initials deleted
WHITE, JON EUBANK MANCHIP	first names spelled out
J.E.M. WHITE	rotated author name
MANCHIP WHITE, J.E.	rotated and partially spelled o
WHI#, JEM	truncated name
WHITE, JO EUB MAN	partially spelled out

Author Name Processor [4]

TABLE 2

Author Name	Address	Age	Book Name	Publisher	Date
ADAMS	xyz	26	a	A	1968
ADAMS	xyz	26	b	B	1972
BROWN	wuv	35	c	A	1950
BROWN	wuv	35	d	A	1957
BROWN	wuv	35	e	B	1958
DOE	rat	40	f	C	1952
⋮	⋮	⋮	⋮	⋮	⋮

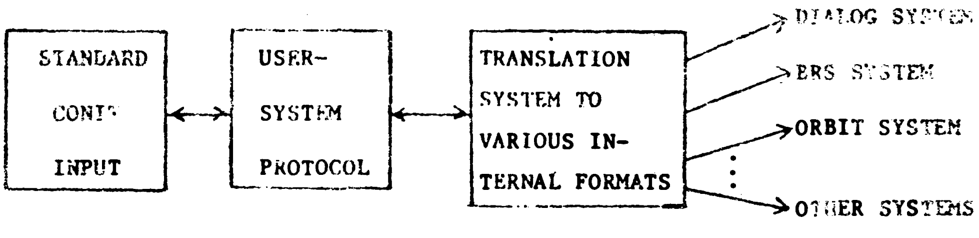
Typical Formatted File

TABLE 3

	relevant	nonrelevant	
term present $x_i=1$	r_i	n_i-r_i	n_i
term absent $x_i=0$	$R-r_i$	$N-R-n_i+r_i$	$N-n_i$
	R	$N-R$	N

Occurrence Characteristics for Term i

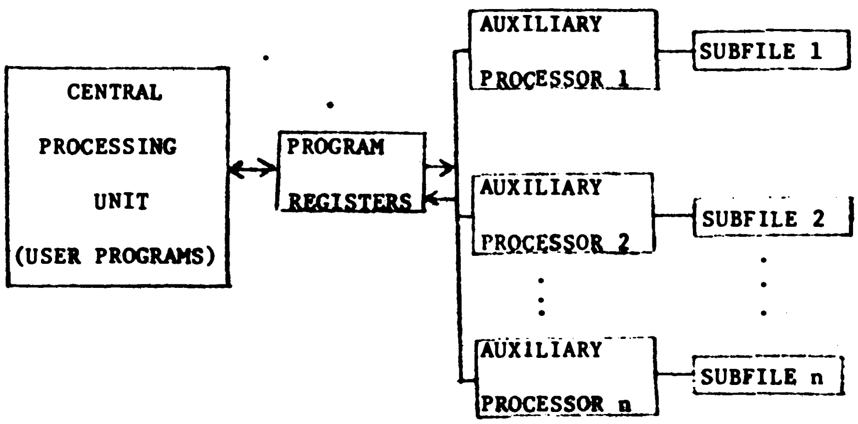
TABLE 4



Simplified Standardization of User-

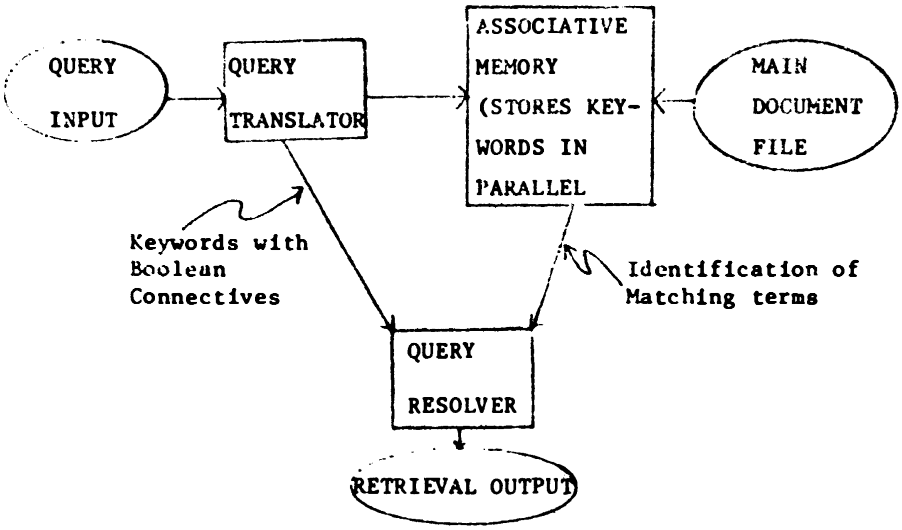
System Interface (CONIT)

FIGURE 1



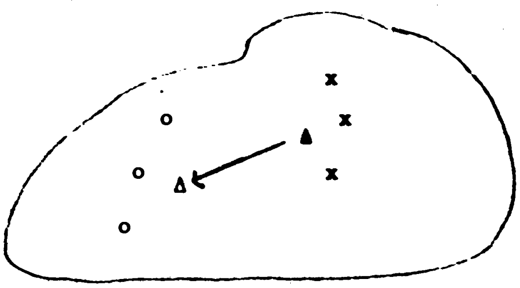
Search Arrangement with Parallel Auxiliary Processors

FIGURE 2



Query-Record Matching with Associative Memory

FIGURE 3



Relevance Feedback

FIGURE 4

x retrieved nonrelevant items
o retrieved relevant items
▲ initial query
Δ reformulated query

Relevance Feedback

FIGURE 4