Retrieving Records from a Gigabyte of Text on a Minicomputer Using Statistical Ranking

Donna Harman and Gerald Candela
National Institute of Standards and Technology, Gaithersburg, MD, 20899

Statistically based ranked retrieval of records using keywords provides many advantages over traditional Boolean retrieval methods, especially for end users. This approach to retrieval, however, has not seen widespread use in large operational retrieval systems. To show the feasibility of this retrieval methodology, research was done to produce very fast search techniques using these ranking algorithms, and then to test the results against large databases with many end users. The results show not only response times on the order of 1 and 1/2 seconds for 806 megabytes of text, but also very favorable user reaction. Novice users were able to consistently obtain good search results after 5 minutes of training. Additional work was done to devise new indexing techniques to create inverted files for large databases using a minicomputer. These techniques use no sorting, require a working space of only about 20% of the size of the input text, and produce indices that are about 14% of the input text size.

Introduction

Statistically based ranked retrieval of records using keywords has been a research success for over twenty years. Experiments have been made involving ranking techniques, and there is general agreement within the research community on the utility of ranking the retrieved records. Although several in-house information retrieval systems have been based on ranking (including Porter & Galpin, 1988; Stibic, 1980; and Brzozowski, 1983), the transfer of this retrieval technique into large operational systems has been very slow. One reason is the general inertia of these large, well-established systems (Radecki, 1988), but another reason is the lack of clear proof that statistical ranking can be done in real time on large databases, and will be accepted by end-users in place of or in addition to Boolean retrieval. The goal of this research was to help establish such proof.

Received June 14; 1989; revised October 20, 1989; accepted October 26, 1989.

The research leading to this article, therefore, was in two major areas. The first area involved producing very efficient indexing and searching algorithms, efficient both in retrieval and indexing time, and in the amount of storage needed for indices. The second area of research involved some preliminary design of interfaces to allow user testing, and then testing against databases such as manuals, legal code books with chapters and sections, and a gigabyte of data containing about 40,000 separate court cases. Here the goal was to examine how well the system served the needs of various user groups.

Past Work in Statistical Ranking

Historically, information searching has been in the form of published catalogs of manually indexed records, and the advent of computers produced multiple efforts at mapping this same system into a faster version of the same concept. Current retrieval systems such as MEDLINE (McCarn, 1980) manually index medical journals and search these indices using Boolean connectors to relate useful terms. Many experiments in automatic indexing in the 1950s and 1960s allowed the terms used in the records themselves to serve as indices to the record, and the effectiveness of this approach was particularly well demonstrated by Cleverdon in his Cranfield experiments (Cleverdon & Keen, 1966). Again the commercial retrieval systems utilized these results by doing Boolean retrieval based on the full text of articles and abstracts, and these systems, such as LEXIS/NEXIS (Mead, 1985), represent the bulk of commercial systems today.

The work in automatic indexing also led to a new type of retrieval methodology based on automatically matching the query terms and the record terms, and ranking the retrieved records based on the "goodness" of that match. The user inputs a simple query such as a sentence or a phrase (no Boolean connectors) and retrieves a list of records ranked in order of likely relevance. One of the early developers of this methodology was Salton in his SMART project (Salton, 1971), and that project showed the feasibility of this approach. For 20 years the information retrieval research community has been expanding this technique using various models, such as probabilistic models, vector space models, and fuzzy logic models, and various matching (similarity) measures and term weighting functions. These experiments have been reported in proceedings of the ACM SIGIR conferences and in several books (Sparck Jones, 1981; Salton & McGill, 1983; VanRijsbergen, 1979). A recent survey (Belkin & Croft, 1987) of these ranking techniques argues their superiority over traditional Boolean retrieval. The commercial retrieval systems have generally not utilized these new methods, with a few notable exceptions (Koll, Noreault, & McGill, 1984).

The present effort to show the large-scale feasibility of this statistical ranking approach uses similarity measures and term weighting functions based on these past experiments. In particular, a ranking weight is assigned to each record based on a formula (see appendix) involving a function of the frequency of query terms in that record and a function of the distribution of the query terms within the database. The frequency of query terms in a record is usually a good indication of the extent to which that record is concerned with those terms. The distribution of a query term within a database is useful in weighting the relative importance of terms in a query. For example the term "computer" in a computer science database is relatively unimportant; in an agricultural database it is probably an important query term. One measure of this relative importance is provided by the inverse document frequency weight (IDF, see appendix for details, Sparck Jones, 1972). The record frequency weight and the IDF weight are combined (and normalized by a function of the length of the record) to form a total record weight for use in ranking. This particular ranking measure was chosen both because of its proven effectiveness over a range of test collections (Salton & McGill, 1983; Harman, 1986) and because of its implementation simplicity. Again, the goal of this research was not to produce a better ranking methodology, but to show that the current laboratory techniques are effective for large-scale record retrieval by end users.

The remainder of the article is in three major sections. First, a section on indexing is presented, showing the methods used to create the inverted files needed for searching. The next section is on the search engine itself, detailing the search techniques and presenting results from experiments done to enhance response times. The final major section describes the user testing, discussing both user reaction to the prototype retrieval system and response times against large databases.

Indexing Methodology and Experiments

Traditional automatic indexing methods producing inverted files can be divided into two or three sequential steps. First, the input text must be parsed into a list of words with their location in the text. This is usually the most time consuming and storage consuming operation in indexing. Second, this list must then be inverted, from a list of terms in location order to a list of terms sorted for use by the search engine (usually sorted into alphabetical order, with a list of all locations attached to each term). This process can involve multivolume tape sorts for very large databases. The optional third step is the postprocessing of these inverted files, such as for adding term weights, or reorganizing the files.

Indexing very large databases using traditional indexing methods on a minicomputer presents many problems. Minicomputers normally cannot handle multivolume tape sorts. This requires that the inverting step be done without sorting for large databases. Additionally, the amount of storage available for processing is relatively small on a minicomputer and this can limit the maximum work space, possibly to a size on the order of the text size itself or two gigabytes total storage for a gigabyte input text file.

The new indexing method developed is a two-step process that does not need the middle sorting step. The first step produces the basic inverted file, and the second step adds the term weights to that file and reorganizes the file for maximum efficiency (Fig. 1).

The creation of the basic inverted file avoids the use of an explicit sort by using a right-threaded binary tree (Knuth, 1973). As each term is identified by the text parsing program, it is looked up in the binary tree, and either is added to the tree, along with related data, or causes tree data to be updated. The data contained in each binary tree node is the current number of postings (the number of records containing one or more instances of the term) and the storage location of the postings list for that term. The postings are stored as multiple linked lists, one linked list for each term, with the lists stored in one large file. Each element in the linked postings file consists of a record id (the location of a given term), the term frequency in that record, and a pointer to the next element in the linked list for that given term. By storing the postings in a single file, no storage is wasted, and the files are easily accessed by following the links. As the location of both the head and tail of each linked list is stored in the binary tree, the entire list does not need to be read for each addition, but only once for use in creating the final postings file (step two).

Note that both the binary tree and the linked postings list are capable of further growth. This is important in indexing large databases where data is usually processed from multiple separate files over a short period of time. The use of the binary tree and linked postings list could be considered as an updatable inverted file. Although these structures are not as efficient to search, this method could be used for creating and storing supplemental indices for use between updates to the primary index. The use of the linked posting list as an updating strategy for inverted files has been suggested earlier by van Rijsbergen (vanRijsbergen, 1976).

The binary tree and linked postings lists are saved for use by the term weighting routine (step two). This routine walks the binary tree and the linked postings list to create an alphabetical term list (dictionary) and a sequentially stored postings file. To do this, each term is consecutively

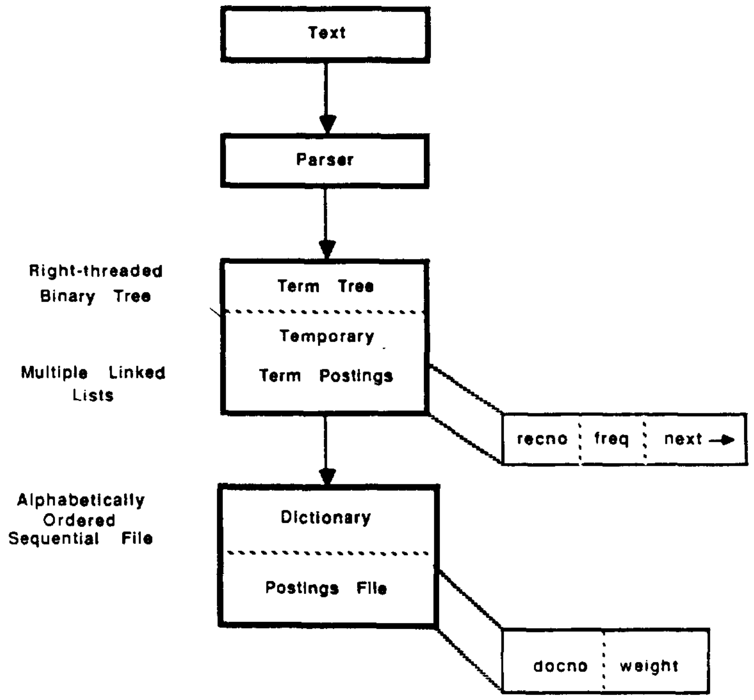


FIG. 1. Flowchart of new indexing method.

read from the binary tree (this automatically puts the list in alphabetical order), along with its related data. A new sequentially stored postings file is allocated, with two elements per posting. The linked postings list is then traversed, with the frequencies being used to calculate the term weights. The last step writes the record ids and corresponding term weights to the newly created sequential postings file. Note that this final file need only contain two elements per posting as no link pointer is needed. Using a sequentially stored posting list in place of a linked list saves storage and speeds access time, as input can be read in multi-record buffers, one buffer usually holding all records for a given term. These sequentially stored postings files could not be created in step one because the number of postings is unknown at that point in processing, and input order is text order, not inverted file order.

The final index files therefore consist of the dictionary and the sequential postings file. Each element of the dictionary contains the term, its IDF weight, the number of postings of the term in the entire database, and the location of its postings in the postings file. Each element in the postings file contains a record id and the term weighting for the given term in that record.

The term weighting is the measure of that term's importance in the given record, which is precalculated and stored to allow speed in the search (ranking) process. The precalculation allows the ranking process to simply sum the weights of all selected terms in a record to get the total weight for the record, which is then used in ranking the records. The term weight is therefore calculated for each term within a record using the formula in the appendix, minus the summation (which is done in the search process itself).

Precalculating the weights, however, precludes the ability to easily change ranking algorithms. Because the results from step one of indexing are saved, the ranking algorithm can be changed by redoing the second step in indexing and changing the term weights.

Table 1 gives some statistics showing the differences between the old and the new indexing schemes. The old indexing scheme refers to a version of a traditional index-

TABLE 1. Indexing statistics.

Text Size (megabytes)	Indexing Time (hours)		Working Storage (megabytes)		Index Storage (megabytes)	
	old	new	old	new	old	new
1.6	0.25	0.50	4.0	0.7	0.4	0.4
50	8	10.5	132	6	4	4
359	—	137	—	70	52	52
806	—	313	—	163	112	112

ing method in which records are parsed into a list of words within record locations, the list is inverted by sorting, and finally the term weights are added.

Note that the size of the final index is small, only 8% of the input text size for the 50 megabyte database, and around 14% of the input text size for the court cases (the 359 megabyte database is one of the subsets of the full court cases). This size remains constant when using the new indexing method as the format of the final indexing files is unchanged. The working storage (the storage needed to build the index) for the new indexing method is not much larger than the size of the final index itself, and substantially smaller than the size of the input text. However, the amount of working storage needed by the old indexing method would have been approximately 933 megabytes for the 359 megabyte database, and over 2 gigabytes for the 806 megabyte database, an amount of storage beyond the capacity of many environments. The new method takes more time for the very small (1.6 megabyte) database because of its additional processing overhead. As the size of the database increases, however, the process time has an $n \log n$ relationship to the size of the database. The traditional method contains a sort which is $n \log n$ (best case) to n squared (worst case), making processing of the very large databases likely to have taken longer using the old method, and considerably longer if a tape sort was required because of the large amount of working storage.

Further attempts to reduce final storage using the new method involved using a bitmapping strategy instead of the postings files. This strategy (Lefkovitz, 1975) creates a bitmap for each term instead of a postings list. The bitmap contains 1 bit per record, or 1400 bits for the Cranfield collection, and these bits are set to one only if the given term appears in the record corresponding to the bit. Whereas bitmapping strategies require constant storage per term (1 bit for each record in the database), for terms with many postings this can result in significant storage savings. Bitmapping schemes carry a major drawback, however, in that no term weighting can be done on a per record basis as only the presence or absence of a term is known. This can lead to a drop in search performance. Further discussion of this topic is found in the searching methodology section.

Search Methodology and Experiments

One way of using an inverted file to produce statistically ranked output is to first retrieve all records containing the search terms, then use the frequency information for each term in those records to compute the total weight for each of those retrieved records, and finally sort those records (Harman, 1988 et al.). The search time for this method is directly related to the number of retrieved records and becomes prohibitive when used in large databases.

This process can be made much less dependent on the number of records retrieved by using a method developed by Doszkocs for CITE (Doszkocs, 1982). In this method, a block of storage was used as a hash table to accumulate the total record weights by hashing on the record id into unique "accumulator" addresses. This makes the searching process less dependent on the number of retrieved records—only the sort for the final set of ranks is affected by the number of records being sorted.

This was the method chosen for the current search engine (Fig. 2). A block of storage containing an "accumulator" for every unique record id is reserved, usually on the order of 300 Kbytes for the large databases. A binary search for a given query term is executed against the dictionary, and the address of the postings list for that term is returned. The postings list is then read, with the term weight for each record id being added to the contents of its unique accumulator. These term weights were precalculated during indexing to allow this method of processing. As each query term is processed, its postings cause further additions to the accumulators. When all the query terms have been handled, accumulators with non-zero weights are sorted to produce the final ranked record list.

Table 2 illustrates the timing improvements using the very small Cranfield collection of 1400 abstracts, with the times shown for processing the entire query collection of 225 queries. These times are only approximate. The "old" system described in the beginning of the search section had processing times ranging from 3½ hours to 30 minutes, depending on the searching parameters used, and was designed for flexibility rather than speed. The time given for the "old" system is the time using searching parameters matching those being used in the "new" system.

The time has been cut drastically, even though more records are retrieved (there is a smaller commonword list in the new system and therefore more terms are used for retrieval). Note that retrieval results are unaffected by this change in search methodology in that the retrieved records are ranked equivalently by both systems.

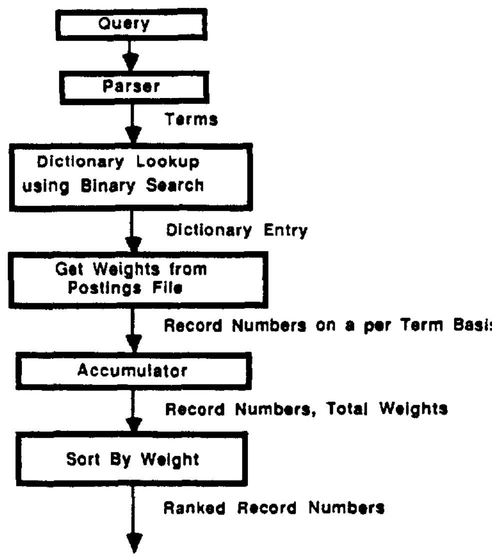

Query
Parser
Terms
Dictionary Lookup using Binary Search
Dictionary Entry
Get Weights from Postings File
Record Numbers on a per Term Basis
Accumulator
Record Numbers, Total Weights
Sort By Weight
Ranked Record Numbers

FIG. 2. Flowchart of search engine.

TABLE 2. Search times.

Retrieval System	Number of Queries	Search Time	Total Records Retrieved
Old	225	~210 minutes	165752
New	225	~5 minutes	206646

To find ways of further improving the search times, a series of experiments was run using the Cranfield test collection of 1400 abstracts. The test collection was used because retrieval results could be verified to see if any degradation in performance was found using these methods.

The first set of experiments involved the use of bitmapping. To minimize storage, terms that would have a postings file storage greater than the bitmap storage were bitmapped. All other terms were handled using the normal postings file method. For the Cranfield 1400, this meant all terms having postings of more than 44 records would have a storage savings using bitmapping. This method not only saves indexing storage, but can be used to decrease the number of records sorted (and hence search time). If query terms are processed in order of importance (as measured by their IDF value), then the terms that have term weights (as opposed to bitmaps) can be accumulated as before. Contributions from the higher-frequency bitmapped terms can then be added only to nonzero accumulators, i.e. can be used only to increase the total weight of records already retrieved by weighted terms, not add more records for sorting. Because fewer total records are sorted, the search times can be improved. Table 3 gives the results for the Cranfield collection using this method for bitmapping of terms based on the number of postings for a term. The average precision is a standard performance measure based on the precision of a search (the ratio of the number of relevant records retrieved to the total number of records retrieved) as averaged over ten levels of recall (the ratio of the number of relevant records retrieved to the total number of relevant records) (Salton & McGill, 1983).

As can be seen from the table, bitmapping all terms with more than 44 postings reduces the storage and search time considerably, but also hurts performance significantly. Bitmapping all terms with more than 200 postings still reduces time and storage significantly, but does not significantly hurt retrieval performance. Bitmapping can also be based on a function of the number of postings, such as the IDF of a term. This allows a more general bitmapping specification based on individual collection statistics rather than on absolute numbers. An example of this is shown in the last column of Table 3, where bitmapping is done if the IDF of a term is less than 1/3 the maximum IDF of any term in the database.

A second method to reduce the number of records sorted involved pruning. This method was based on the fact that many records for most queries are retrieved based on matching only query terms of high database frequency (this assumes bitmapping is not being used). These records are still sorted, but serve only to increase sort time, as they are seldom, if ever, useful. These records can be retrieved in the normal manner, but pruned before addition to the retrieved record list (and therefore not sorted). Experiments were run using the Cranfield collection to try this method. The results are shown in Table 4.

If records are pruned based on matching only query terms with an IDF less than 1/2 the maximum IDF of any term in the database, only about 5% of the records retrieved need to be sorted and the processing time is cut nearly in half. However the recall/precision is hurt signifi-

TABLE 3. Bitmapping experiments.

	No Bitmapping	Bitmapped if >44 postings	Bitmapped if >100 postings	Bitmapped if >200 postings	Bitmapped if idf < maxidf/3
Postings storage (bytes)	346424	180594	197463	243946	243313
Total records retrieved	206646	7231	30805	82061	81622
Total search time (225 queries)	5:03	2:36	2:52	3:38	3:33
Average precision	0.391	0.261	0.352	0.385	0.384
Precision change (%)		−33.2	−10.2	−1.7	−1.9

TABLE 4. Pruning experiments.

	No Pruning	Pruning if idf < maxidf/2	Pruning if idf < maxidf/3
Total records retrieved	206646	206646	206646
Total records sorted	206646	11188	78470
Total search time (225 queries)	5:03	2:45	3:36
Average precision	0.391	0.306	0.390
Precision change (%)		−22.0	−0.3

cantly. Because the term distribution in text databases is highly skewed towards low frequency terms (approximating a Zipf distribution (Tague & Nicholls, 1987), pruning at so low a threshold eliminates records having terms with posting counts starting at 52 in this database, and these terms are clearly not high frequency terms. If the pruning requirements are relaxed to prune only records with matching terms having an IDF less then 1/3 the maximum IDF (starting at a posting count of about 200), about 40% of the retrieved records need to be sorted, and the time savings is still significant. The recall/precision is not significantly hurt at this level of pruning, however, and therefore this time savings is not at the expense of performance.

Note that the search time and performance for both bitmapping and pruning at a cutoff of 1/3 the maximum IDF are similar. This is because these two techniques are effectively the same: pruning uses postings to find marginal records, but prunes these records before sorting; bitmapping uses bitmaps to select these records, but does not add them to the record list for sorting because no nonbitmapped terms appear in them. These two timing saving techniques produce similar results for the Cranfield collection, except there are no storage savings using pruning. User testing did show some differences between bitmapping and pruning, however, and these are discussed in the section on user testing. Similar work has been done by Buckley and Lewit (Buckley, 1985), with an extensive investigation into more complex "stopping" conditions for pruning. Perry and Willett (1983) and Lucarella (1988) also discussed efficient ways of handling large collections, including other pruning techniques.

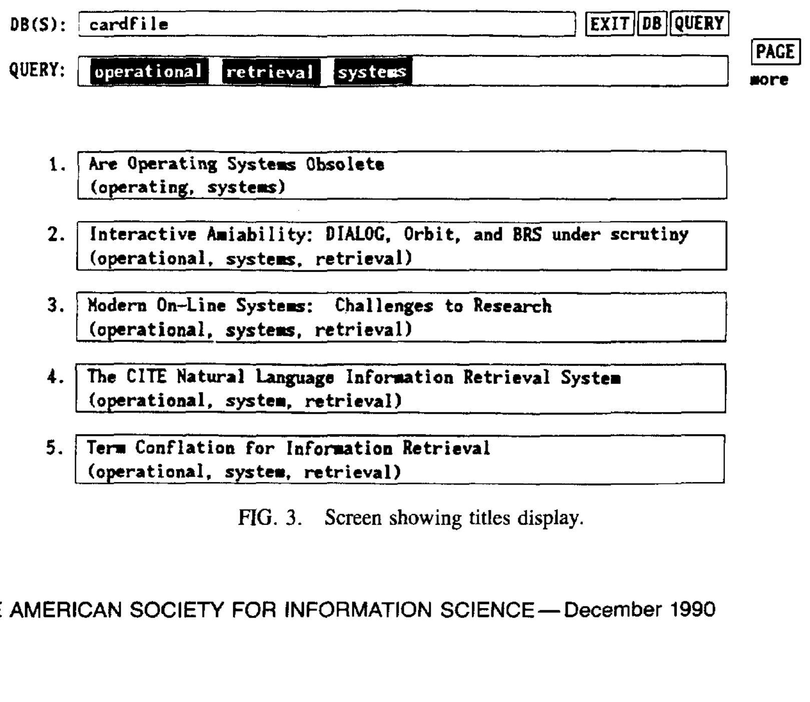
User Testing

The previous section was based on batch processing of a test collection. To test the search methodology online with both end users and large databases, a simple interface needed to be designed. This interface was a mouse-controlled X-Windows interface that was extremely simple to operate, with users starting operations using mouse-selected buttons, inputting a query by typing into one of the windows, and selecting records for display from a list of titles for the retrieved records in ranked order (Fig. 3). The work in the design of this interface and the user testing leading to improvements in the interface are beyond the scope of this paper. Of interest here is how well the search methodology performed for real users.

The results of this testing fall into three areas. First, did the search engine perform fast enough to satisfy online use, and how did the bitmapping and pruning affect the online timing. Second, did the ranking produce satisfactory results for the users, and third, what kinds of problems need to be addressed to produce a truly usable retrieval system.

Three different databases were involved in the testing. The first database was about the size of the Cranfield abstract collection, but consisted of a manual. The manual was relatively nontechnical and was organized into sections and chapters. A record was determined to be equivalent to a paragraph in the manual, because this appeared to be the most useful record size for the end users. This decision caused many short records (Table 5). The second database was a legal code book, again with sections and subsections. Here the language was very technical, and the records were set to be each subsection, again based on user preference. The records were therefore much larger, with many words occurring multiple times within each record. The third database consisted of over 40,000 court cases. This database had a natural division into three smaller databases (this is how it was actually used) with separate indexing for each database, and parallel searching when all three parts of the database were of interest. A record here was set to be a court case. Table 5 shows some basic statistics on these databases. The average number of terms per record includes duplicate terms and is a measure of the record length rather than the number of unique term occurrences.

The number of records and the average number of terms per record is a function of the definition of a record, and



1. Are Operating Systems Obsolete
(operating, systems)

2. Interactive Amiability: DIALOG, Orbit, and BRS under scrutiny
(operational, systems, retrieval)

3. Modern On-Line Systems: Challenges to Research
(operational, systems, retrieval)

4. The CITE Natural Language Information Retrieval System
(operational, system, retrieval)

5. Term Conflation for Information Retrieval
(operational, system, retrieval)

FIG. 3. Screen showing titles display.

TABLE 5. Database statistics.

Size of databases	1.6 meg	50 meg	268 meg	359 meg	179 meg	806 meg
Number of records	2653	6652	9064	17495	11745	38304
Average terms per record	96	1124	4565	3147	2434	3264
Number of unique terms	5123	25129	126624	166802	78781	243470
Average postings per term	14	40	52	59	65	88

this should be based on logical divisions of the data and its expected usage. In particular, the 1.6 megabyte manual was generally searched for single facts, and a paragraph was the logical division, causing many short records. Each court case, however, is considered a logical entity, and this causes very large records. The number of unique terms is relative not only to the size of the database, but to how the database is indexed. The 50 megabyte database, for example, had many numbers indexed because of user requirements. This created many additional unique terms and a larger dictionary, with many terms having a single posting. This problem was magnified in the databases involving court cases, where many unusual formatting characters were not removed, and therefore many single incidences of "misspelled" words were created. Retrieval was not usually affected by this, but the dictionaries were much larger than they normally would be.

The users that tested the system consisted of over 40 people, all proficient in doing research (usually manual research) in the three databases as part of their regular job. About 2/3 of these users had never or seldom used an online retrieval system. Nine of the testers used Boolean retrieval systems regularly, and five others had used Boolean retrieval systems, but not regularly enough to develop expertise. All users brought trial queries with them, often questions they were currently manually researching, or had researched in the past. These queries covered a wide range of the subject areas contained in the databases.

The searching hardware was a minicomputer having six Intel 80386 processors. Except for the parallel processing mentioned above, only one processor was used for a given search. The minicomputer had an internal memory of 16 megabytes, a per processor speed of about 3 to 4 mips, and used an SMD disk interface, with an average access time of 20 milliseconds and an average data transfer speed of 2 Megabytes per second. The software was written in C for a UNIX operating system.

The system was very fast. Initial response time for most queries on the largest databases usually tested (268 megabytes) was on the order of 2 and 1/2 seconds. Table 6 gives the average response time using a test group of user queries for each database.

The response time for the 806 megabyte database assumes parallel processing of the three parts of the database, and would be longer if the database could not be processed in parallel. Bitmapping was not done for all databases, but the pruning showed significant savings, retrieving records from the 806 megabyte database in about 1 and 1/2 seconds.

The response time using pruning is 23% less for the smallest database, going up to a 56% time savings in the 268 megabyte database and yielding a response time of about 1 second. The bitmapping takes longer than the pruning in these large databases because the processing of bitmapped terms takes somewhat longer than the processing of terms with postings, and this effect begins to be noticeable, even though the same number of records is being sorted. Both the bitmapping and pruning were done using a cutoff based on the IDF of a term being less than 1/3 the maximum IDF in the database.

The system was also very effective. Looking only at the test queries, 53 out of 68 queries retrieved at least one relevant record in the top ten, with 25 of these retrieving relevant records as the top record. Other user queries not recorded as test queries worked equally well, and users were very pleased with the results. Most of the users were not experienced searchers but got successful results within 5 minutes after the demonstration because there was no search logic to learn. This ease of use was combined with the natural ability of a ranking retrieval system to handle partial matches; that is, the output can be viewed as a continuous relaxation of a Boolean query. In one testing session users brought in 10 queries that had been previously run against a Boolean system. The system retrieved the single relevant record within the top five records for all 10 queries, whereas the Boolean system had missed the relevant record completely in three of the queries.

Note that the test queries in general had only one relevant record because the users were usually looking for an answer to a question rather than trying to find all possible records that could answer the question. This means that the precision of the system was very high, but the recall is unknown. It is not clear how important recall is in the situation where a user is looking for an answer, a situation well addressed by Cooper (Cooper, 1976). Interestingly, users that were looking for high recall in the test sessions tended to run several queries rather than expect a single query to

TABLE 6. Response time.

Size of database	1.6 meg	50 meg	268 meg	806 meg
Number of queries	13	38	17	17
Average number of terms per query	4.1	3.5	3.5	3.5
Average number of records retrieved	797	2843	5869	22654
Average response time per query in seconds	0.38	1.2	2.6	4.1
Average response time (bitmapping)	0.29	0.72	1.8	—
Average response time (pruning)	0.28	0.58	1.1	1.6

provide all the answers. This may be a pattern learned in doing manual research, but also might be more effective in online retrieval where many relevant records cannot be retrieved based on the initial queries but need query modification (Harman, 1988).

The queries that did not retrieve well fell into three categories. Most of the queries that failed tried to retrieve a record in which the words in the relevant record were simply not those in the query. This is the most common problem for all full-text retrieval systems where there is no manual indexing, thesaurus, or relevance feedback to provide alternative search terms. Some queries retrieved their relevant records after rank ten because the query was very general, but clearly the user had something more specific in mind when selecting the single relevant record. The rest of the queries that performed poorly did so for a set of reasons which are listed below.

- The system provided automatic stemming which expanded the queries to include all possible term variants according to the Lovins stemming algorithm. This was usually very effective. In several queries, however, stemming hurt performance badly.
- Both the bitmapping and pruning eliminated records containing only high frequency terms. A few queries contained only high frequency terms, and therefore the relevant record(s) for these queries contained only multiple high frequency terms. Both pruning and bitmapping eliminated these records.
- Some queries contained numbers and "common" words that were critical to retrieval.
- In the largest databases the ability to retrieve using phrases improved performance greatly. This retrieval involved a secondary pass through the ranked records, showing the user only those records containing the given phrase. The 17 test queries run against the court cases had 6 queries that retrieved relevant records at ranks greater than 10. Allowing the same words to be grouped into phrases not only caused 5 of those queries to retrieve the relevant record in the top five, but improved performance for most of the 17 queries.

Whereas the above problems do not significantly hurt averages in a test collection environment, it is very important that any operating retrieval system handle any legitimate query well. Stemming problems can be resolved by allowing a user to prevent a word from being stemmed. The pruning or bitmapping needs to be modified to provide special processing for queries with only high frequency words. This tends to favor the pruning mechanism over bitmapping because both postings and bitmaps must be stored in order to handle these queries, and this eliminates the storage advantage of bitmapping. The project is moving in the direction of providing these and other features necessary for an operating retrieval system. One of these other features is the expansion of the secondary pass methodology used in handling phrases to allow field restrictions, term adjacency, and Boolean connectors. These features seldom proved to be useful in a ranking environment, but seemed to be considered necessary features by the four user testers who regularly conduct searches as intermediaries rather than end users. If these features can be provided without hurting response time or increasing the index size, they will make the system more acceptable to traditional searchers.

Conclusion

Several new techniques were investigated in the course of this research in addition to extensive user testing. A new method of building inverted files on a minicomputer was developed that used no sorting and needed working storage space of only about 20% of the size of the input text file. The size of the final index was also kept small, on the order of 14% of the input text size.

A search engine was built based on a technique that allowed the response time to be less dependent on the number of records retrieved. The basic search engine required less than 5 minutes to process the entire Cran-field 1400 test collection, with 225 queries, and around 4 seconds to search a query against an 806 megabyte text database.

Two additional time-saving methods, bitmapping and pruning, were tried. Both kept the number of records being sorted for the ranking to a minimum. The pruning technique cut the average response time to about 1 and 1/2 seconds for the 806 megabyte database. None of these time-saving techniques significantly hurt the retrieval performance of the system.

The user testing showed the power of a ranking environment very well. Not only did the system retrieve relevant records in the top ten shown to the user for most queries, but users were generally very enthusiastic about the ease of use.

In summary, the prototype retrieval system using statistically-based ranked retrieval of records not only performed at speeds equivalent to or better than current operational systems, but provided results equivalent to or better than those provided by these systems. The ranking schema allows novice users the opportunity to be successful searchers with minimal training, and, as important, with no learning curve for searchers only using the system occasionally. This is in sharp contrast to Boolean based retrieval systems where continual use is necessary to obtain consistently good results. Clearly this technology, once available only in laboratory settings, is ready for widescale implementation in large operational systems.

Acknowledgments

Many thanks to all the reviewers of this paper for both excellent suggestions and additional pointers to new references.

Appendix. Ranking Using the IDF Weighting Measure

$$\text{rank}_j = \sum_{k=1}^{Q} \frac{(\log_2 \text{Freq}_{jk} \times \text{IDF}_k)}{\log_2 M_j}$$

where Q = the number of terms in the query

Freq_{jk} = the frequency of query term k in record j

IDF_k = the inverse document frequency weight of term k for a given database

M_j = the total number of significant terms (including duplicates) in the record j (length factor)

$$\text{IDF}_k = \log_2 \frac{N}{\text{Num}D_k} + 1$$

where N = the number of records in the database

$\text{Num}D_k$ = number of records in the collection that contain one or more instance of term k

References

Belkin, N. J., & Croft, W. B. (1987). Retrieval techniques. In Williams, Martha (Ed.), *Annual Review of Information Science and Technology.* New York: Knowledge Industry Publications, Inc.

Brzozowski, J. P. (1983). MASQUERADE: searching the full text of abstracts using automatic indexing. *Journal of Information Science, 6,* 67–74.

Buckley, C., & Lewit, A. (1985). Optimization of inverted vector searches. In *Proceedings of the eighth international conference on research and development in information retrieval,* (pp. 97–110) Montreal, Canada.

Cleverdon, C. W., & Keen, E. M. (1966). *Factors determining the performance of indexing systems, Vol. 1: Design, Vol. 2: Test results.* Aslib Cranfield Research Project, Cranfield, England.

Cooper, W. S. (1976). The paradoxical role of unexamined documents in the evaluation of retrieval effectiveness. *Information Processing and Management, 12,* 367–375.

Doszkocs, T. E. (1982). From research to application: The CITE natural language information retrieval system. In Salton, G. and Schneider, H. J. (Eds.), *Research and development in information retrieval.* Berlin: Springer, pp. 251–262.

Harman, D. (1986). An experimental study of factors important in document ranking. In *Proceedings of the 1986 ACM conference on research and development in information retrieval,* Pisa.

Harman, D., Benson, D., Fitzpatrick, L., Huntzinger, R., & Goldstein, C. (1988). IRX: An information retrieval system for experimentation and user applications. In *Proceedings of RIAO '88,* Boston, pp. 840–848.

Harman, D. (1988). Towards interactive query expansion. In *Proceedings of the 11th international conference on research and development in information retrieval,* Grenoble, France, pp. 321–331.

Koll, M., Noreault, T., & McGill, M. (1984). Enhanced Retrieval Techniques on a Microcomputer. In *Proceedings of the national online meeting,* New York, N.Y., pp. 165–170.

Knuth, D. E. (1973). The art of computer programming. Reading, MA: Addison-Wesley Publishing Company.

Lefkovitz, D. (1975). The large data base file structure dilemma. *Journal of Chemical Information and Computer Sciences, 15,* 14–19.

Lucarella, D. (1988). A document retrieval system based on nearest neighbor searching. *Journal of Information Science, 14,* 25–33.

McCarn, D. B. (1980). MEDLINE: An introduction to on-line searching. *Journal of the American Society of Information Science, 31,* 181–192.

Mead Data Central. (1985). *LEXIS/NEXIS Quick Reference Guide,* Dayton, OH.

Perry, S. A., & Willett, P. (1983). A review of the use of inverted files for best match searching in information retrieval systems. *Journal of Information Science, 6,* 59–66.

Porter, M. F., & Galpin, V. (1988). Relevance feedback in a public access catalogue for a research library: Muscat at the Scott Polar Research Institute. *Program, 22,* 1–20.

Radecki, T. (1988). Trends in research on information retrieval—The potential for improvements in conventional boolean retrieval systems, *Information Processing and Management, 24,* 219–227.

Salton, G. (1971). *The SMART retrieval system—Experiments in automatic document processing.* Englewood Cliffs, NJ: Prentice-Hall.

Salton, G., & McGill, M. (1983). *Introduction to modern information retrieval.* New York: McGraw-Hill Book Company.

Sparck Jones, K. (1972). A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation, 28,* 11–20.

Sparck Jones, K. (1981). Information retrieval experiment. London: Butterworths.

Stibic, V. (1980). Influence of unlimited ranking on practical online search strategy. *Online Review, 4*(3), 273–279.

Tague, J., & Nicholls, P. (1987). The maximal value of a zipf size variable: Sampling properties and relationship to other parameters. *Information Processing and Management, 23,* 155–170.

van Rijsbergen, C. J. (1976). File organization in library automation and information retrieval. *Journal of Documentation, 32,* 294–317.

van Rijsbergen, C. J. (1979). *Information retrieval,* 2nd edition, London: Butterworths.