Parallel Text Search Methods[*]

Gerard Salton
Chris Buckley

87-828

April 1987

Department of Computer Science
Cornell University
Ithaca, New York 14853-7501

[*]This study was supported in part by the National Science Foundation under grants IST 83-16166 and IST 85-44189.

Parallel Text Search Methods

G. Salton and C. Buckley[*]

Abstract

An evaluation of recently proposed parallel text search methods does not support the notion that the parallel methods provide large-scale gains in either retrieval effectiveness or efficiency, compared with alternative available search strategies using serial processing machines.

1. Boolean Text Retrieval Methods

Since punched card days, text searches have been conducted automatically by comparing the content attributes, or terms, assigned to the text items with the term combinations included in the query statements. In conventional retrieval environments, the search requests are normally represented by *Boolean statements*, consisting of search terms interrelated by the Boolean operators *and, or* and *not*, and the retrieval system is designed to select those stored items which are identified by the exact combination of search terms specified in the available queries. Thus given a four-term query statement such as "(A *and* B) *or* (C *and* D)", the retrieved items will contain either the term pair A and B, or the pair C and D, or both pairs. The terms characterizing the stored texts may be assigned manually by

[*]Department of Computer Science, Cornell University, Ithaca, NY 14853.
This study was supported in part by the National Science Foundation under grants IST 83-16166 and IST 85-44189.

trained personnel; alternatively, so-called automatic indexing methods may be used where the term assignment is handled automatically. In some systems, the content analysis, or indexing operation, is entirely avoided by using for content identification certain words contained in the texts of the corresponding documents. [1-3]

In operational retrieval situations, the queries are generally submitted using console terminal equipment, and responses are received while the users wait at the terminals. [4-5] In many cases, the size of the searchable collection may be quite large, often exceeding a million or more items. Responses are nevertheless obtained in real time because auxiliary indexes are used which specify for each search term the lists of document identifiers containing each particular term. Thus given lists of document references corresponding to terms A and B, respectively, responses to queries such as (A *and* B), or (A *or* B), are easily obtained by manipulating the record identifier lists stored in the auxiliary index.

Table 1 shows a typical example, consisting of two ordered lists of document references for the two terms A and B. A single, ordered and merged (A,B) list can be constructed by successive comparison of pairs of items from the separate A and B lists, and traversal of the merged list then provides responses to query (A *and* B) consisting of all duplicated items, or query (A *or* B), consisting of the unique items from the merged list. The merged (A,B) list and the set of query responses are shown in the middle of Table 1 for the original sample A and B lists.

The retrieval performance of the existing Boolean text search systems varies widely depending on the type of search request actually submitted and on the homogeneity of the collection being searched. [6-7] When very specific queries are processed, only a few items may be obtained, but most of the retrieved items are likely to be useful. On the other hand, when the queries are broadly formulated, many more stored items are retrieved, including both relevant items as well as extraneous ones that may not prove useful to the requestor. When the stored collection covers a well-defined subject area, the retrieval performance is generally better than for collections covering many different topic areas.

The conventional retrieval environment has been widely accepted, because the Boolean formulations can be used to express term relationships, such as for example synonym relations identified by *or* operators ("minicomputers *or* microcomputers *or* handheld calculators"), and term phrases specified by *and* operators ("information *and* retrieval"); furthermore fast responses are obtained even for very large document collections. Unfortunately, the Boolean environment also proves disadvantageous in many cases, most importantly because nonexpert users find it difficult to generate effective query formulations that produce the proper amount of output and the expected proportion of relevant materials. Furthermore, in the Boolean environment it is difficult to distinguish important query terms from less important ones by assigning weights or other importance parameters to the query terms. In the conventional Boolean system all terms are treated as equally important and all retrieved documents are assumed to be equally useful. The order in which the retrieved items are presented to the user population is thus arbitrary, and does not normally correspond to any presumed order of importance of the items. [8-10]

Alternative retrieval models have been considered in an attempt to overcome the limitations of the conventional Boolean approach. The best known of these is the vector processing system.

2. The Vector Processing System

With the introduction of large-scale computers in the 1950's, it became possible to use more sophisticated text processing operations than was possible in the earlier punched card environment. In particular, complex query-document comparison operations could be implemented reflecting the degree of similarity between query formulations and stored texts, and replacing the earlier simple matches between assigned query and document terms. In the *vector space* model, the information items are identified by sets of attributes, or terms, as in the Boolean system. However, instead of assuming that all terms are equally important or valuable, *term weights* are used to distinguish the degree of importance of the terms. Assuming that the system supports t distinct terms usable for content identification, a given document D_i

can then be represented as a t-dimensional vector of pairs of the form

$$D_i = (d_{i1}, w_{d_{i1}};\ d_{i2}, w_{d_{i2}};\ \ldots;\ d_{it}, w_{d_{it}}) \tag{1}$$

where d_{ij} represents the jth term assigned to document D_i, and $w_{d_{ij}}$ is the corresponding term weight. In principle, all t terms could appear in each term vector, a 0 weight being used for terms that are not present in a given document, and larger weights between 0 and 1 designating the terms actually assigned to the items. [11-12]

In the vector processing system, the Boolean query formulations are replaced by weighted term sets of the form

$$Q = (q_1, w_{q1};\ q_2, w_{q2};\ \ldots;\ q_t, w_{qt}) \tag{2}$$

where once again t distinct terms are assumed to exist, and a 0 weight is used for terms that are absent. When the stored texts and user information requests are represented as weighted term vectors, a global, composite measurement can be used to represent the similarity between a query-document pair, based on the weights of the corresponding matching terms. A typical, composite vector similarity measure that has been widely used is the *cosine measure*, representing the cosine of the angle between query Q and document D_i, considered as vectors in t-space:

$$similarity\ (Q, D_i) = \frac{\sum_{j=1}^{t} w_{qj} \cdot w_{dij}}{\sqrt{\sum_{j=1}^{t} (w_{qj})^2 \cdot \sum_{j=1}^{t} (w_{dij})^2}} \tag{3}$$

Expression (3) ranges from 0 when there are no terms in common between documents and queries to 1 when all terms match and are fully weighted. The illustration of Fig. 1 shows that the similarity between Q and D_1 as measured by the cosine ratio y/z is much larger than the similarity between Q and D_2, measured by x/z. D_1 is correspondingly closer to Q in the space than D_2. By arranging the stored texts in decreasing order according to the size of the query-document similarity, a ranked retrieval strategy is obtained which retrieves the stored items in presumed decreasing order of importance. The ranking feature also serves to control the size of the retrieved output, because the retrieval operation can simply be discontinued after the user has seen a sufficient number of items. The global comparison of weighted term vectors which forms the basis for the vector processing retrieval model can of course be used not only in text processing, but is directly extendable to any pattern matching environment where attribute vectors representing collections of objects can be compared and ranked.

Three main strategies are available for actually carrying out the collection search in a vector processing environment:

a) A *sequential search* can be used to compare the query with each stored document in turn, computing in each case the corresponding query-document similarity coefficients. Because each document must be individually matched with the query in a sequential search, this solution is usable only in special circumstances when the files are very small.

b) In some situations, a classified or *clustered* file organization is available where records covering the same subject matter are grouped into affinity classes. This is true notably of many library files where the documents are grouped according to library classification numbers. For clustered collections, fast search strategies are available in which the search effort can be concentrated in the most productive document clusters. [13-15]

c) Auxiliary index files, similar to those previously described for the Boolean search system, can be used to provide lists of record identifiers corresponding to each of the search terms. In a vector processing environment, it is useful to include the term weights in the index lists because the query-document similarity factors can then be computed directly from the stored index information.

If real-time responses are needed and a clustered file is not available, the use of indexed searches provides a reasonably alternative. Augmented term lists similar to those shown at the bottom of Table 1 are then used, where each document identifier is followed by the weight of the given term in that document. Thus in the example of Table 1, term A carries a weight of 0.2 in document D_2, and a weight of 0.5 in D_{15}, while term B has a weight of 0.1 in D_5 and a weight of 0.9 in D_{10}. If the similarity measure of expression (3) is used to compute the query-document similarities, the index list for term j would contain the weighting factor $w_{d_{ij}} / \sqrt{\sum_{k=1}^{t} (w_{d_{ik}})^2}$ following the entry for document D_i. Since the weights of the query terms can be separately specified, the complete expression (3) is then easily generated from the stored index information.

3. The Smart Retrieval System

The Smart system (System for the Manipulation and Retrieval of Texts) was designed in the early 1960's as a prototype text processing and retrieval system. The Smart system is based on the vector processing model and uses query-document similarity computations similar to those of expression (3). Successive implementations were prepared in the 1960's on IBM 7090 and 7094 machines, in the 1970's on IBM 360 and 370 equipment, and most recently on a variety of machines running UNIX such as the DEC VAX 11/780 and Microvax computers. [16-19] The Smart system uses automatic indexing techniques to generate the query and document vectors from available query and document texts. [20-21]

The value of a term attached to a document or query is assumed to depend directly on the importance of the term in the document text--for example on the frequency of occurrence of the term in the text--and indirectly on the number of documents in the collection to which the term is attached. That is, the best terms--those which can most easily distinguish the documents to which they are attached from the remainder of the collection--are those occurring frequently in individual text items, but rarely in the remainder of the collection. Accordingly, a typical term weighting strategy usable in the Smart environment is

$$w_{d_{ij}} = tf_{ij} \cdot \log \frac{N-f_j}{f_j} \qquad (4)$$

where tf_{ij} is the frequency of occurrence of term j in document D_i, f_j is the number of documents in the collection to which term j is attached and N is the collection size. [22,23] When the weighting function of expression (4) is used, the weight of term j in document D_i increases as the term frequency tf_{ij} increases, and as the collection frequency of the term f_j decreases.

A dynamic query improvement technique, known as *relevance feedback* is utilized in the Smart system which produces new, improved query formulations based on information derived from documents retrieved in earlier search operations. [24-25] Specifically, indexed searches are carried out that provide a ranked output display of the retrieved items in decreasing similarity order with the respective queries. Some of the displayed items are then identified by the user as being useful or relevant to his information needs, or on the contrary as extraneous or not relevant. The query is then automatically reformulated to render it more similar to the items previously designated as relevant, and less similar to the items identified as nonrelevant. Relevance feedback can be adapted to Boolean query environments . [26,27] However, the process is especially attractive for vector queries where the relevant (or nonrelevant) document vectors are simply added to (or subtracted from) the original query statements. The following feedback equation in vector notation is in fact optimal under reasonable conditions: [24]

$$Q_{i+1} = Q_i + \alpha \sum_{Rel} D_i - \beta \sum_{Nonrel} D_i \ . \qquad (5)$$

Here Q_{i+1} is the new query statement obtained from the previous query formulation Q_i by adding the terms from all relevant documents with a multiplication factor α, and subtracting the terms from the nonrelevant items with a multiplicative factor of β. The feedback operation of expression (5) may change the weight of existing query terms that are included in certain previously retrieved relevant or nonrelevant documents. Alternatively, new terms derived from the previously retrieved documents may be added to the query with appropriate weighting factors.

The feedback operation is graphically illustrated in the "vector space" picture of Fig. 2 where the documents and queries are represented by the tips of the corresponding vectors. The distance between two points in the picture is inversely related to the similarity between the corresponding vectors. An original query (open triangle) is shown in Fig. 2 together with three retrieved documents (x) that are assumed to be relevant. When the terms from these relevant items are added to the query, a new query designated by a closed triangle in Fig. 2 is formed which is much more similar to the previously retrieved items than the original query. The new query may then retrieve additional relevant items that would have been missed by the original query.

The available evidence indicates that the relevance feedback operation provides substantial gains in retrieval performance. [24-25, 27] Because the relevant documents used for query modification will necessarily be retrieved once more by the feedback query with much improved ranks, the feedback run will always appear to produce better output than the original run, even when the user does not actually obtain any new relevant items that have not previously been seen. In evaluating the relevance feedback operation one must therefore be careful to subtract the so-called ranking effect caused by the improvement in the retrieval ranks of previously retrieved relevant items that were used for query modification purposes.

Table 2 illustrates the ranking effect by displaying an assumed document ranking for both initial and feedback searches. Three relevant documents labelled R_1, R_2 and R_3 in Table 2 are initially retrieved with retrieval ranks 1, 3, and 7. When the terms from these documents are incorporated in the query, the ranks of these items improve from 1, 3, and 7, to 1, 2, and 3, respectively. The retrieval effectiveness of the run in Table 2 is measured by the well-known *recall* and *precision* parameters, reflecting respectively the proportion of relevant items retrieved, and the proportion of retrieved items that are relevant. The recall (R) may thus be computed as the number of retrieved relevant items divided by the total number of relevant items in the collection. Correspondingly, the precision (P) is computed as the number of retrieved, relevant items divided by the total number of retrieved items.

In the example of Table 2, the assumption is that the collection contains 10 relevant documents in all. In these circumstances the recall increases by 1/10 each time a relevant item is found. The performance figures on the right side of Table 2 indicate that at every retrieval level following the first retrieved item, the feedback run outperforms the initial run in both recall and precision even though in fact the feedback run retrieves no additional relevant items up to the fifth retrieved item.

Various solutions offer themselves for measuring the true advantage provided by the relevance feedback process. [28-30] One possibility is the so-called *residual collection* system where all items previously seen by the user (whether relevant or not) are simply removed from the collection, and both the initial and any subsequent searches are evaluated using the reduced collection only. This depresses the absolute performance level in terms of recall and precision, but maintains a correct relative difference between initial and feedback runs. The residual collection evaluation is used to evaluate the relevance feedback searches examined later in this note.

4. Parallel Computations in Text Processing

Various attempts have been made in recent years to introduce parallelism in the text processing and information retrieval areas. The most obvious approach consists simply in overlapping, or performing simultaneously, processes that use different machine resources and can therefore be carried out independently of each other. For example, incoming Boolean queries can be parsed, that is decomposed into into clauses, at the same time as incoming document texts are analyzed and indexed using dictionary consulting methods. Alternatively, retrieved document citations can be displayed while simultaneously rephrasing the queries using relevance feedback operations. This type of coarse-grained parallelism is especially attractive when the available operating system supports the corresponding procedures. Under UNIX, user queries that share certain query terms are usefully processed together, because the document lists corresponding to the common query terms are then extracted from the auxiliary indexes and transferred into buffer memory only once instead of many times. Obviously, no special equipment is needed to use this type of parallelism.

An alternative possibility for improving the efficiency of operations consists in introducing special-purpose devices designed to provide fast execution of particular operations that must be performed frequently, and that prove particularly time-consuming under normal circumstances. An obvious candidate is the list merging operation needed to process Boolean query clauses such as (A *and* B), or (A *or* B). When this operation is carried out sequentially, the number of pairwise comparisons between elements of the two lists is equal to the sum of number of elements on both lists. The list merging operation can, however, be speeded up considerably by using a number of comparison units in parallel, each capable of comparing two input elements and of identifying the smaller of the two elements. Thus N/2 comparison units may be used for lists of length N/2 to compare respectively the first entries from each of the two lists (D_2 and D_5 in the illustration of Table 1), the next two elements from each list (D_{15} and D_{10}), the third elements (D_{23} and D_{15}), and so on until the final two elements (D_{148} and D_{190}) are allocated to the last comparison unit. By suitably feeding the output of the first stage of the comparison operation to a second stage using additional compare units, and continuing on to additional comparison stages, a *merging network* can be built which produces a merged list in substantially fewer than N operations. In particular, when two lists of N/2 entries are handled, the numer of comparison stages needed is $(1 + \left\lceil \log_2 N/2 \right\rceil)$ instead of N as before, and the number of required comparison units is $N + N/2 \log_2 N/2$. [31] Parallel list merge methods have been implemented in various experimental information retrieval systems. [32-33]

The parallel list merge system is useful in retrieval systems based on auxiliary term index lists. In the vector processing model, parallel operations may also be used to carry out the vector manipulation operations.

Thus the vector similarity computations of expression (3) could in principle be speeded up by performing all the multiplication operations on the different vector elements in parallel, followed by a parallel addition of the individual partial products. The introduction of parallel vector operations substantially enhances the speed of query-document comparisons, as well as the document-document comparisons used in collection clustering, and the vector additions needed in relevance feedback (expression (5)).

Special backend vector processing machines, known as *array processors*, have been developed capable of performing vector operations in parallel. The array processors offer a number of parallel functional units, including for example parallel adders, multipliers, and parallel logical units. In addition, pipelining is used to overlap the components of the individual operations. Array processors (AP's) can be attached to front-end "host" machines, in which case they rely on the host for loading the needed information into the AP memory before the parallel operations are carried out, and for storing the results produced by the AP operations.

A host-array processor complex has been used experimentally to implement various information retrieval operations in the vector processing model. [34-35] A typical allocation of operations for the host-array processor combination is shown in Table 3. In the example of Table 3 a cluster search is carried out which consists in first comparing the query with the "centroid vectors" that characterize the contents of the individual document clusters, followed by a comparison of the query with the document vectors located in designated document clusters only.

In performing the vector similarity computations, the use of array processors provides a two-fold advantage: the computations themselves are carried out more rapidly; in addition, the cost of running the array processor is much smaller than the cost of computing on the host machine. The available evaluations indicate, however, that the pure speed advantage of the array processor is substantially reduced by the need to transfer data back and forth between host and backend processors. For a sample computation of 400 query-document similarity measurements, 145 milliseconds were required by a stand-alone IBM 370 system. This was reduced to a total of 13 milliseconds when the AP was in use, plus 45 milliseconds of AP time. [34] Considering the fact that a great variety of different operations must be carried out in text retrieval environments, most of which are not improvable by array processing devices, a speed advantage of 3 to 1 for vector similarity computations alone may not be sufficiently attractive in practical situations.

It was mentioned earlier that in operational retrieval systems the search file may consist of several million items. The complication and expense of introducing and maintaining the auxiliary term indexes designed to provide fast answers in response to incoming search requests may then be avoided by using specialized back-end search processors designed to carry out sequential searches on small subsections of the data files. In particular, if n such back-end search processors were available, the record file could be divided into n pieces, all of which would be searched in parallel by the individual back-end machines. When the number of back-end processors is sufficiently large, an exhaustive search of all records in the stored files can be carried out that nevertheless provides real time responses to the user population. [36-39]

A typical multiprocessor arrangement of the type described earlier is represented in Fig. 3. This configuration introduces substantial conceptual simplifications in search applications because rapid answers are now obtainable without auxiliary file clustering or index maintenance operations. On the other hand, the record files must be efficiently apportioned to the back-end machines, and control procedures of substantial complexity may be needed to coordinate the operations of the parallel processors. Although various design proposals exist for so-called cellular, parallel processors, these machines have not so far been applied in operational situations.

In recent years, the back-end processor philosophy has been extended by greatly increasing the number of back-end machines and having each processor carry out the same operations on the data under its control. Conceptually, it is then possible to use as many processors as there are documents in the collection, and hence to carry out the search for the whole collection in a single global operation performed by all machines in parallel. Such a data-driven organization simplifies the control operations and makes it possible to use conceptually very simple individual processor designs. The so-called Connection Machine (CM) has been proposed as an implementation of the back-end search machine. [40-41] The relevant details of the text search operations using the CM hardware organization are covered in the next section.

5. Use of the Connection Machine for Text Searching.

The Connection Machine (CM) is a complex of up to 65,000 (2^{16}) interconnected parallel processors. Each processor contains an operations unit capable of performing various bit-level operations, such as for example the addition of bits or the logical-*or* of bits, as well as a small internal memory capable of storing 4096 bits. As the name indicates, the various processors are connected with each other, and data can be moved from one processor to another under the control of stored routing information. In principle, each processor executes the same operation at the same time on the different data stored in the individual processors. In practice, control data may be stored to distinguish the processors from each other and to limit the various operations to certain chosen processors only.

Before outlining the search and retrieval operations on the Connection Machine, it is useful to describe briefly the methods used to store the documents in the CM processors. In principle, it would be useful to store document vectors of the type shown in expressions (1) and (2), consisting of sets of weighted terms. Assuming that a document is identified by 100 terms of 8 characters each, 6400 bits are needed to the store the terms alone. Additional space would have to be allocated for the term weights. Since each CM processor carries only 4096 bits of storage, including 1000 or so bits taken up by control information, it is obvious that the standard term vector representation would have to be spread across several processors. Such an organization complicates the query-document comparison operations, and limits the number of documents that will fit into the available machine.

For this reason, the Connection Machine designers propose a *bit-string* representation to encode each document. In particular, each term assigned to a document is represented by a binary word of n bits (a bit string of length n) with k particular bit positions ($k < n$) being set to 1, and the remaining n-k bits set to 0. Documents identified by a single term are represented directly by the bit string for the corresponding term. Documents identified by more than one term are represented by the superimposed bit vectors for the individual terms; that is, a given bit position in the n-dimensional bit vector representing the document is set equal to 1 whenever that bit position is set equal to 1 in the coded representation of any one of the terms.

The bit-string representation for documents, also known as a document *signature*, actually goes back some 40 years to punched card days when a punched card was prepared for each document, and notches placed along the card edge were used to encode the terms assigned to the documents. A sample document card with 24 punch positions along the top edge is shown in Fig. 4. The assumption in the example is that each term assigned to a document is represented by four chosen notch positions, and that the card edge is appropriately notched when a term is assigned to a document. The notches corresponding to several different terms are superimposed as shown in the example of Fig. 4. In punched card days, a needle technique would be used for retrieval: specifically, to find documents identified by terms A and B, needles would be inserted at the appropriate punch positions into the stack of document cards, and cards with notches cut out in all the needle positions would drop down into a box. [42] In modern bit-string matching systems, the signature for the query terms can be compared directly with the document signature, and documents whose signatures cover the query term signature (in the sense that each 1 in the query term signature is matched by a 1 in the corresponding position in the document signature) can be retrieved. A typical query signature and matching document bit string are shown in the example of Fig. 5.

The superimposed coding system can cause false retrievals when the superposition of several terms produces patterns that are identical to the codes of other assignable terms. For example, if the term "zebra" is encoded in positions 3, 15, 20, and 21 in the example of Fig. 4, documents about "zebras" are necessarily retrieved in answer to queries about "information retrieval". The probability of false retrievals can however be minimized by an appropriate choice of the length n of the bit vectors, and of the number k of bit positions which are set equal to 1 for each encoded term. In particular, it is known that the false retrieval probability is smallest when half of the bit positions in a bit vector are equal to 1. [43,44]

Because the bit string representation of documents and queries saves a great deal of storage space compared with the weighted term vector representations of expressions (1) and (2), document signature representations have been proposed for the implementation of modern text message and document retrieval systems. [45-47] In the Connection Machine implementation, bit strings of length 1024 bits are suggested, and up to 35 terms are encoded into each bit string by setting 10 specified bit positions equal to 1 for each assigned term. For documents represented by 100 terms, 3 different bit strings of 1024 bits each are needed, and these can still be included comfortably in the available 4096 bits of memory available in each processor. This makes it possible in principle simultaneously to compare up to 65,000 document signatures stored in 65,000 different processor memories with each incoming query.

With the exception of the bit vector representation and the absence of individually stored term weights, the retrieval algorithm envisaged for the Connection Machine is closely patterned after the Smart system methodology: [40,41]

a) An indexing process carried out in the host machine is used to reduce the original document texts to a set of terms; the elimination of common words included in a special dictionary is a principal component of the indexing step.

b) A binary document signature is prepared by superimposing the hashed bit representations of the terms.

c) Vector queries (rather than Boolean statements) are used and a binary signature is separately prepared for each term included in a query.

d) The document vectors are loaded into the Connection Machine and a parallel comparison is carried out between each query term and all stored documents; that is, if a query includes m terms, m global similarity computations are needed, each one comparing one of the query terms with all stored documents.

e) A query-document similarity, or *document retrieval value*, is computed for all documents whose signatures cover any of the query term signatures. Since individual term weights are not stored, the document retrieval value used in the CM implementation is computed simply as $\sum_j 1/log\, f_j$ where f_j is the collection frequency of term j (the number of documents to which term j is assigned), and the summation ranges over all document terms j that match the given query terms.

f) A global maximum operation available in the Connection Machine is used to obtain a ranked retrieval of the top 15 or 20 documents in decreasing order of the sum of the inverse collection frequencies of all matching query terms.

g) A *relevance feedback* step is also available in which the user designates some of the retrieved documents as relevant, and a new query is constructed by taking the sets of terms included in the designated relevant documents and including them in the query. Since document term weights are not used in the CM, each new query term j will once again be weighted according to its inverse collection frequency 1/log f_j. The original relevance feedback equation (5) which increases or decreases the individual query term weights is thus replaced by the formulas $Q_{new} = \bigcup_{Relevant\ D_i} D_i$, or $Q_{new} = Q_{old}\ or \bigcup_{\mathrm{Rel}\ D_i} \mathrm{D}_i$.

In evaluating the CM retrieval methodology, two main questions must be considered: the first relates to the speed advantage of the CM compared with the speed of more conventional retrieval methods using sequential or indexed searches; the second concerns the retrieval effectiveness of the CM methodology, that is, the ability to retrieve what is wanted and to reject what is extraneous. Earlier text retrieval evaluations have confirmed the importance of sophisticated term weighting systems capable of assigning different degrees of importance to the terms assigned to distinct documents. This accounts for the use of the individual term frequencies tf_{ij} of term j in documents D_i as illustrated earlier in the term weighting formulas of expression (4). [21-23] Furthermore, when the documents contain varying numbers of terms, it is usually important to incorporate length normalization factors in the query-document comparisons, such as those used in the denominator of the cosine function of expression (3). Without length normalization, documents with more terms receive higher retrieval values and hence have a better chance of being retrieved than documents with fewer terms.

Neither term frequency weights, nor length normalization are available in the CM methodology. Nevertheless, it has been claimed that "unstructured text retrieval (can be performed on the CM) achieving 80 percent recall rates compared to 20 percent for conventional systems". [48] Without attempting to interpret this type of statement--any retrieval system can obviously always reach a perfect 100 percent recall by simply retrieving everything in the collection--it may nevertheless be useful to assess the actual performance of the CM procedures for text retrieval. This is done in the next section.

6. Evaluation of the CM Methodology

Descriptions are available of the performance of a Connection Machine with 16,000 processors, using two search queries with 200-term queries. [40,41] The assumption made in these tests was that each query term occurs in one percent of the collection, that is, in 160 documents. Assuming that the document signatures are already stored in the processor memories and that data transfers are therefore not required to load information from the host machine to the CM, the time needed by the CM to compare a 200-term query with the 16,000 stored document is 40 milliseconds; another 20 milliseconds are needed to retrieve the 20 top documents in decreasing query-document order. (The 40 ms search time actually suffices to process 65,000 documents by using four CM modules each capable of handling 16,000 documents.)

In interpreting the CM test results, Waltz remarks that 265.8 million instructions must be executed by a serial machine performing a sequential search of each of 65,000 documents for a 200-term query (or equivalent, 66.45 million instructions would be needed for a sequential search of 16,000 documents). This implies that the 40 ms search time of the CM can be attained by a serial machine only if the machine operates at a rate of 6645 million instructions per second (MIPS) for 65,000 documents or equivalently 1661 MIPS for 16,000 documents. [41] While the conclusion may be formally correct, it is also quite misleading because no one in his right mind would in fact perform serial searches for text collections comprising tens of thousands of documents.

The relevant timing data for indexed document searches of the type actually performed in practice are shown in Table 4. Using the assumptions made for the CM searches with respect to collection size and query length, about 140 thousand instructions are required for an indexed search by a serial machine. In these circumstances, a SUN3 computer operating at a speed of 3.5 MIPS performs the search operations as rapidly as the Connection Machine, while a dedicated VAX operating at one MIPS requires a search time of 140 ms. In the SMART system environment, a multipurpose VAX computer is used that performs other text processing tasks in addition to collection searching. In these circumstances, the auxiliary term index is not kept in internal storage, but the term lists for the query terms are loaded into memory from a disk store as needed. When disk transfers are included, the response time on a standard VAX is about 1.5 seconds for 16,000 documents using a 200-term query.

In interpreting these figures, it must be remembered that in a text search application, the retrieval process will be controlled by the query analysis and query formulations speed (which are measured in seconds rather than milliseconds even in the CM), as well as by the speed at which the user can absorb the output information. In particular, a few minutes are normally required by users in order to read the displayed information, and to submit the relevance assessments needed for query reformulation in a relevance feedback process. In these circumstances, even the 1.5 second response time that includes disk transfer operations will appear adequate. In any case, a dedicated SUN3 computer with sufficient internal memory for the storage of the term indexes is as efficient as a comparable CM in this application. Surely the SUN3 or VAX equipment will be far less expensive to acquire and run than the parallel Connection Machine. One must conclude that from the point of view of retrieval speed, the CM does not provide any essential advantage for the text retrieval application.

Consider now the problem of retrieval effectiveness. The CM methodology uses only inverse collection frequency weights attached to query terms. This must be compared with the use of alternative term weighting systems that are available in vector processing environments. Three types of term weighting components must be distinguished:

a) a term frequency component which is based on the frequency of occurrence of a term in a given document or query;

b) a collection frequency component which is based on the number of documents in a collection to which a term is attached;

c) and a term weight normalization component which insures that the lengths of query or document vectors are equal.

Table 5 gives typical assignments for the various term weighting components. By using a variable to designate each weighting component, a complete term weight specification is expressible as a triple, covering the term frequency, inverse collection frequency, and vector normalization components respectively. The computation of the document retrieval values is completely specified by using the product of two adjacent triples, such as "bnn·btn", where the first triple refers to the weighting method used for document terms, the second triple refers to query term weights, and the retrieval value of a document is computed by using a simple inner product function consisting of the sum of the product of corresponding weights for matching query and document terms. (Terms absent from a given query or document vector are assumed to carry a 0 weight.)

A typical similarity computation is shown in Table 6 for the ntc (document) · atn (query) weighting system. In this case a normalized term frequency times inverse collection frequency known as (tf · idf) weighting scheme is used for document terms, and an enhanced term frequency times inverse document frequency system characterizes the query terms. The enhanced frequency component used for query terms in Table 6 is designed to insure that these weights lie in the range between 0.5 and 1.0

The retrieval performance of the CM search methods is reflected in the evaluation data of Tables 7 to 11. In each case four different document collections in four different subject areas, known as MED, CACM, CISI and INSPEC, are used, ranging in size from about 1,000 documents for MED to about 13,000 for INSPEC. The performance of a given method is evaluated by using a single number representing the average search precision obtained at three different recall values of 0.25, 0.50, 0.75 recall respectively. The average precision values at these three recall levels are further averaged over a number of search requests available for each collection, ranging from 30 queries for MED to 77 queries for INSPEC. The performance figures of Table 7 to 11 thus represent averages of averages that are somewhat difficult to interpret in absolute terms. However, the average precision values for different retrieval runs do of course accurately reflect the relative differences between the retrieval effectiveness of the different methods.

The output of Table 7 confirms that the inverse collection frequency weighting used in the CM document ranking system is in fact useful in retrieval. In this case a totally unweighted retrieval system (bnn·bnn) is compared with the CM method (bnn·btn) using unweighted terms attached to documents, but inverse collection weights (idf) for the query terms. The product (bnn·btn) represents the CM retrieval order according to the sum of the inverse collection frequencies of matching terms in queries and documents. The output of Table 7 shows that the idf weighting system provides improvements in average search precision ranging from 20 percent for the CISI collection to over 60 percent for INSPEC.

The experiments of Table 8 demonstrate that the CM base case (bnn·btn) is not optimal. When weighted document terms are used instead of unweighted document terms, substantial additional improvements in retrieval effectiveness are produced ranging from 8 percent in average precison for the MED collection to over 40 percent for INSPEC. As expected, the term weighting system produces relatively less improvement for collections with a high performance standard such as MED, than for the others where more room is left for performance increases. In Table 8, the idf term weighting system is maintained throughout for the query terms, but the document term weights are changed from binary (that is, unweighted) to term frequency (tf) and (tf·idf) weights. The document term weights are not accessible to the CM process because of the bit-string document representation.

The output of Table 9 carries the term weighting system one step further by introducing better term weighting methods (than simple idf) also for the query terms. When tf·idf weights are used for both query and document terms, the improvement in retrieval effectivness over the standard CM base case ranges from over 10 percent for the MED collection to nearly 80 percent for CISI. The (ntc·atn) weighting used in the right-most column of Table 9 was described earlier in the example of Table 6. The evaluation data of Tables 7 to 9 demonstrate that while the CM query weighting system in inverse collection frequency is beneficial compared to a totally unweighted retrieval method, much greater improvements can be produced in a vector processing environment such as Smart, that is not based on a binary document signature representation.

The use of discrimination factors distinguishing relatively good terms assigned for content representation from relatively poorer ones is even more crucial in a relevance feedback situation. In the CM environment where document term weights are not available, the relevance feedback step consists simply of *adding* all terms included in documents designated as relevant to the original query to form a new query. Alternatively, the original query could be *replaced* by the set of terms included in the designated relevant documents. Such an approach treats each document term as equally useful for query formulation purposes. The evaluation output of Table 10 shows that the CM relevance feedback process does not work well in practice.

In Tables 10 and 11, the CM base case (bnn·btn) is used for the initial run throughout. The top 15 documents retrieved by the CM base run are examined for relevance, and the queries are appropriately reformulated. A new (relevance feedback) run is then performed with the altered query whose performance can be compared with that of the original run. As described earlier, the advantages of the feedback run cannot be properly evaluated if the documents used in the feedback operation are maintained in the system. A *residual collection* evaluation is therefore used in Tables 10 and 11 which subtracts the 15 items examined by the user from the collection for evaluation purposes. This accounts for the fact that the CM base run of Table 10 has a lower overall performance than the CM base case of Tables 8 and 9, since the top 15 documents retrieved in response to the queries are now missing in each case.

The true improvement obtainable by the relevance feedback process incorporated into the Connection Machine system is shown in Table 10. As the figures indicate, the feedback process used in the Connection Machine produces a performance degradation ranging from over 30 percent for CISI to over 60 percent for INSPEC. Obviously, it is not possible to use all document terms indiscriminately for query reformulation purposes, as suggested by the CM designers. [40]

The output of Table 11 demonstrates the usefulness of the more sophisticated relevance feedback methods developed in the Smart system environment. The second column of Table 11 uses the basic CM relevance feedback method for query reformulation purposes by constructing a feedback query with idf term weights. This query is then however matched against properly weighted document vectors (tf·idf normalized). Once again the document term weights are essential: the feedback operation now produces improvements of up to 60 percent for most collections.

To obtain substantial benefits from the feedback operation, it is necessary to use more sophisticated query reformulation processes. The selective query term alteration system of expression (5) introduced in the Smart system is used for the runs presented in columns 3 and 4 of Table 11. Here the weights of old query terms are selectively raised or lowered in accordance with the weights of these terms in the relevant or nonrelevant previously retrieved documents. In addition, new terms not originally present may be added to the query. In the last two feedback runs of Table 11, ntc (that is, normalized term frequency times inverse collection frequency) weights are used for query reformulation purposes. Moreover, a feedback method introduced by Ide [29] is used which consists in utilizing *all* previously retrieved relevant documents for query reformulation as well as *one* previously retrieved nonrelevant item (the top nonrelevant previously retrieved item).

The third column of Table 11 uses the Smart feedback process with selective weight modification and (tf·idf) document term weights for query-document comparison purposes. An additional normalization factor is used in the run shown in column 4 of Table 11 for the document vectors. The results of Table 11 show that the performance deterioration noted earlier for the CM feedback methodology, is now replaced by very considerable increases in performance for the properly reformulated and re-weighted queries, ranging from an improvement of about 130% for the MED collection to about 170 percent for CISI and INSPEC. Additional feedback iterations would no doubt increase the performance still further. [24-27]

Conclusion

There is no question about the potential usefulness of the "memory based" reasoning methods proposed by Waltz and others, and of the importance of the global comparisons and subsequent rankings of weighted attribute vectors representing item properties or item content. [40,41] This type of approach is surely usable in many types of pattern matching situations, such as data base retrieval, medical diagnosis, and character recognition. Furthermore, the search time obviously decreases when parallel search methods are used such as those proposed for the Connection Machine. The experiments described in this study show, however, that the method of implementation is important and that parallelism *per se* does not in itself improve search and retrieval output.

In the text retrieval application, the search speed is not of primary importance, since user response time will in fact control the total speed of operations. Furthermore, indexed searches are usable with modern serial machines whose response speed is completely competitive with that of the proposed parallel search equipment. In text retrieval it is also essential that important terms be distinguished from less important ones. Without sophisticated term weighting systems, acceptable retrieval output is not obtainable either for initial searches or for the feedback search iterations. If the memory units attached to the small parallel processing units cannot accommodate sophisticated term weight, the losses in retrieval effectiveness of several hundred percent compared with alternative nonparallel vector matching processes are so large, that the parallel search methods will not prove attractive in practice.

In the end, the potential use of the CM, or of other parallel search devices, depends on cost factors. When relatively small data collections are processed as in certain SDI (selective dissemination of information) systems, and core swapping is not required too frequently, and when discriminating attribute vectors are usable in each parallel processing unit, a fast parallel matching system should certainly be considered. For general commercial use with large collections of several million items these conditions cannot be met. Since indexed search methods are currently available that can rapidly eliminate most extraneous items from consideration, and that can also accommodate term weights or other term discrimination factors, the global vector matching systems developed over the past 25 years for serial computing devices appear more attractive in most existing text processing situations.

References

[1] F.W. Lancaster, Information Retrieval Systems: Characteristics, Testing and Evaluation, Second Edition, J. Wiley and Sons, New York, 1979.

[2] C.J. van Rijsbergen, Information Retrieval, Second Edition, Butterworths, London, 1979.

[3] G. Salton and M.J. McGill, Introduction to Modern Information Retrieval, McGraw Hill Book Company, New York, 1983.

[4] C.T. Meadow and P.A. Cochrane, Basics of On-line Searching, J. Wiley and Sons, New York, 1981.

[5] W.M. Henry, J.A. Leigh, L.A. Tedd, P.W. Williams, On-Line Searching -- An Introduction, Butterworths, London, 1980.

[6] D.C. Blair and M.E. Maron, An Evaluation of Retrieval Effectiveness for a Full-Text Document Retrieval System, Communications of the ACM, 28:3, March 1985, 289-299.

[7] G. Salton, Another Look at Automatic Text-Retrieval Systems, Communications of the ACM, 29:7, July 1986, 648-656.

[8] G. Salton, E.A. Fox, H. Wu, Extended Boolean Information Retrieval, Communications of the ACM, 26:11, November 1983, 1022-1036.

[9] W.S. Cooper, Exploiting the Maximum Entropy Principle to Increase Retrieval Effectiveness, Journal of the ASIS, 34:1, January 1983, 31-39.

[10] C. Cleverdon, Optimizing Convenient On-Line Access to Bibliographic Databases, Information Services and Use, 4, 1984, 37-47.

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

[12] G. Salton, C.S. Yang and A. Wong, A Vector Space Model for Automatic Indexing, Communications of the ACM, 18:11, November 1975, 613-620.

[13] K. Sparck Jones, Some Thoughts on Classification for Retrieval, Journal of Documentation, 26:2, June 1970, 89-101.

[14] W.B. Croft, A Model of Cluster Searching Based on Classification, Information Systems, 5:3, 1980, 189-195.

[15] G. Salton and A. Wong, Generation and Search of Clustered Files, ACM Transactions on Database Systems, 3:4, December 1978, 321-346.

[16] G. Salton and M.E. Lesk, The Smart Automatic Document Retrieval System - An Illustration, Communications of the ACM, 8:6, June 1965.

[17] G. Salton, The Evaluation of Automatic Retrieval Procedures - Selected Text Results Using the Smart System, American Documentation, 16:3, June 1965, 209-222.

[18] G. Salton, Smart, Encyclopedia of Computer Science and Technology, J. Belzer, A.G. Holzman, A. Kent, editors, 13, Marcel Dekker Inc., New York, 1979, 137-172.

[19] G. Salton, editor, The Smart System - Experiments in Automatic Document Processing, Prentice Hall Inc., Englewood Cliffs, NJ, 1971.

[20] G. Salton, C.S. Yang and A. Wong, A Vector Space Model for Automatic Indexing, Communications of the ACM, 18:11, November 1975, 613-620.

[21] G. Salton, A Theory of Indexing, Regional Conference Series in Applied Mathematics No. 18, Society for Industrial and Applied Mathematics, Philadelphia, PA, February 1975.

[22] K. Sparck Jones, A Statistical Interpretation of Term Specificity and its Application in Retrieval, Journal of Documentation, 28:1, March 1972, 11-21.

[23] G. Salton, C.S. Yang and C.T. Yu, A Theory of Term Importance in Automatic Text Analysis, Journal of the ASIS, 26:1, January-February 1975, 33-44.

[24] J.J. Rocchio Jr., Relevance Feedback in Information Retrieval, Scientific Report ISR-9, Harvard Computation Laboratory, Cambridge, MA, August 1965, also Chapter 14 in The Smart Retrieval System - Experiments in Automatic Document Processing, G. Salton, editor, Prentice Hall Inc., 1971, 313-323.

[25] G. Salton, Relevance Feedback and the Optimization of Retrieval Effectiveness, Scientific Report ISR-12, Department of Computer Science, Cornell University, Ithaca, NY, June 1967, also Chapter 15 in The Smart Retrieval System - Experiments in Automatic Document Processing, Prentice Hall Inc., 1971, 324-336.

[26] V. Vernimb, Automatic Query Adjustment in Document Retrieval, Information Processing and Management, 13:6, 1977, 339-353.

[27] G. Salton, E.A. Fox and E. Voorhees, Advanced Feedback Methods in Information Retrieval, Journal of the ASIS, 36:3, May-June 1985, 200-210.

[28] H.A. Hall and N.H. Weiderman, The Evaluation Problem in Relevance Feedback Systems, Report No. ISR-12 to the National Science Foundation, Section XII, Department of Computer Science, Cornell University, Ithaca, NY, June 1967.

[29] E. Ide, New Experiments in Relevance Feedback, Report No. ISR-14 to the National Science Foundation, Section VIII, Department of Computer Science, Cornell University, Ithaca, NY, October 1968, also Chapter 16 in The Smart Retrieval System - Experiments in Automatic Document Processing, G. Salton, editor, Prentice Hall Inc., Englewood Cliffs, NJ, 1971, 337-354.

[30] Y.K. Chang, C. Cirillo and J. Razon, Evaluation of Feedback Retrieval Using Modified Freezing, Residual Collection and Test and Control Groups, Chapter 17, in The Smart Retrieval System - Experiments in Automatic Document Processing, G. Salton, editor, Prentice Hall Inc., 1971, 355-370.

[31] D.E. Knuth, The Art of Programming, Vol. 3, Searching and Sorting, Addison Wesley Publishing Company, Reading, MA, 1973, 224-230.

[32] W. H. Stellhorn, An Inverted File Processor for Information Retrieval, IEEE Transactions on Computers, C-26:12, December 1977, 1258-1267.

[33] L.A. Hollaar and W.H. Stellhorn, A Specialized Architecture for Textual Retrieval, AFIPS Conference Proceedings, 46, AFIPS Press, Montvale, NJ, 1977, 697-702.

[34] G. Salton and D. Bergmark, Parallel Computations in Information Retrieval, Proc. CONPAR 81, Lecture Notes in Computer Science No. 111, W. Handler, editor, Springer Verlag, Berlin 1981, 328-343.

[35] D. Bergmark and A. Hanushevsky, Document Retrieval: A Novel Application for the AP, FPS User's Group Meeting, Los Angeles, CA, 1980.

[36] S.Y.W. Su, Cellular Logic Devices: Concepts and Applications, Computer, 12:3, March 1979, 11-25.

[37] C.P. Copeland, G.J. Lipovski, and S.Y.W. Su, The Architecture of CASSM: A Cellular System for Nonnumeric Processing, Proc. of the First Annual Symposium on Computer Architecture, Association for Computing Machinery, NY, December 1973, 121-125.

[38] P.J. Sadowski and S.A. Schuster, Exploiting Parallelism in a Relational Associative Processor, Proc. of the Fourth Workshop on Computer Architecture for Nonnumeric Processing, Association for Computing Machinery, New York, August 1978, 99-109.

[39] S.A. Schuster, H.B. Nguyen, E.A. Ozkarahan and K.C. Smith, RAP2 - An Associative Processor for Data Bases and its Applications, IEEE Transactions on Computers, Vol. C-28:6, June 1979, 446-458.

[40] C. Stanfill and B. Kahle, Parallel Free-Text Search on the Connection Machine System, Communications of the ACM, 29:12, December 1986, 1229-1239.

[41] D.L. Waltz, Applications of the Connection Machine, Computer, 20:1, January 1987, 85-97.

[42] C.N. Mooers, Zatocoding Applied to Mechanical Organization of Knowledge, American Documentation, 2:1, Winter 1951, 20-32.

[43] C.S. Roberts, Partial Match Retrieval via the Method of Superimposed Codes, Proc. IEEE, 67:12, December 1979, 1624-1642.

[44] J.L. Pfaltz, W.J. Berman and E.M. Cagley, Partial Match Retrieval Using Indexed Descriptor Files, Communications of the ACM, 23:9, September 1980, 522-528.

[45] D. Tsichritzis and S. Christodoulakis, Message Files, ACM Transactions on Office Information Systems, 1:1, January 1983, 88-89.

[46] P.A. Larson, A Method for Speeding up Text Retrieval, Database, 15:2, Winter 1984, 19-23.

[47] C. Faloutsos and S. Christodoulakis, Signature Files: An Access Method for Documents and its Analytical Performance Evaluation, ACM Transactions on Office Information Systems, 2:4, October 1984, 267-288.

[48] K.A. Frenkel, Evaluating the Massively Parallel Machines, Communications of the ACM, 29:8, August 1986, 752-758.

Term A	:	$\{D_2, D_{15}, D_{23}, D_{89}, D_{123}, D_{140}, D_{148}, \ldots\}$
Term B	:	$\{D_5, D_{10}, D_{15}, D_{50}, D_{90}, D_{123}, D_{190}, \ldots\}$
Merged (A,B)	:	$\{D_2, D_5, D_{10}, D_{15}, D_{15}, D_{23}, D_{50}, D_{89}, D_{90}, D_{123}, D_{123}, D_{140}, D_{148}, D_{190}, \ldots\}$
Retrieved Items for (A *and* B)		$\{D_{15}, D_{123}\}$
Retrieved Items for (A *or* B)		$\{D_2, D_5, D_{10}, D_{15}, D_{23}, D_{50}, D_{89}, D_{90}\ D_{123}, D_{140}, D_{145}, D_{190}, \ldots\}$
Augmented Term A List	:	$\{D_2,0.2; D_{15},0.5; D_{23},0.1; D_{89},0.9; D_{123},1.0; D_{140},0.2; D_{148},0.7; \ldots\}$
Augmented Term B List	:	$\{D_5,0.1; D_{10},0.9; D_{15},0.7; D_{50},0.2; D_{90},0.3; D_{123},0.2; D_{190},0.1; \ldots\}$

List Manipulations in Indexed Searches

Table 1

Initial Document Ranking		Document Ranking after Feedback		Performance			
				Initial		Feedback	
				R	P	R	P
1	R_1 $\longrightarrow$	1	R_1	1/10	1	1/10	1
2	x	2	R_2	1/10	1/2	2/10	1
3	R_2	3	R_3	2/10	2/3	3/10	1
4	x	4	x	2/10	2/4	3/10	3/4
5	x	5	new Rel	2/10	2/5	4/10	4/5
6	x	6	x	2/10	2/6	4/10	4/6
7	R_3	7	x	3/10	3/6	4/10	4/7
⋮		⋮		⋮			

Relevance Feedback Illustration

(R_1, R_2, R_3 are relevant items used for feedback;
R recall = number of relevant retrieved/total relevant;
P precision = number of relevant retrieved/total retrieved)

Table 2

Host Machine	Array Processor (AP)
1. User types in query; query vector is generated and sent to array processor	1. Idle (centroid vectors characterizing cluster structure already stored in AP)
2. Idle	2. Compare query vector with stored centroid vectors and identify best clusters
3. Document vectors located in the clusters most similar to the query are transfered to AP	3. AP starts comparison of query vector that are with some document vectors
4. Search results are obtained from the AP and corresponding document citations are retrieved from the files	4. Query-document comparisons are carried out, and identifiers for most highly matching documents are sent to host
5. Document citations are presented to user and query is reformulated for relevance feedback	5. AP is initialized for a new search

Typical Collection Search Using Host-Array Processor Combination

Table 3

Query Processing

200 terms for query
each term occurs in 1% of collection
16,000 documents

Sequential Processing (using term index)

200 terms × 160 documents/term
× 4 instructions for inner loop
× 1.1 outer loop overhead = 140,800 instructions

Time for Document-Query Comparisons

1. Connection Machine (no disk transfers)*	40 ms
2. SUN 3 (8 Mbyte, 3.5 MIPS)	40 ms
3. VAX 11/780 (8 Mbyte, 1 MIPS)	140 ms
4. VAX 11/780 (multipurpose machine; term index stored on disk)	1,500 ms = 1.5 sec.

Sample Timing for Query-Document Comparisons
(* data from reference [41])

Table 4

Term Weighting Components
(first triple covers document, second triple covers query)

i) left most component covers term frequency
(frequency of occurrence of term in a document:tf)

b	:	1.0
n	:	tf
a	:	$0.5 + 0.5 \frac{tf}{\max tf}$ (max tf in document vector)

ii) middle component covers collection frequency
(number of documents in collection to which a term is assigned:f)

n	:	1.0
t	:	log N/f (inverse collection frequency (idf) component)

iii) third component covers document length normalization

n	:	1.0
c	=	$1/\sqrt{\sum_i (w_{ij})^2}$ (document vectors normalized to maximum length of 1)

Term Weight Construction

Table 5

$$sum(D_i,Q) = \sum_{j=1}^{t} \{ [\overset{n}{(tf_{d_{ij}})} \cdot \overset{t}{(log \frac{N}{f_j})} \cdot \overset{c}{(\frac{1}{\sqrt{\sum_{j=1}^{t} tf_{ij}^2 \cdot (log \frac{N}{fj})^2}})}]$$

$$\cdot [\underset{a}{(0.5 + 0.5 \frac{tf_{g_j}}{\max tf_g})} \underset{t}{(log \frac{N}{fj})} \cdot \underset{n}{1}] \}$$

tf_{qj}	term frequency of term j in Q
$tf_{d_{ij}}$	term frequency of term j in D_i
f_j	collection frequency of term j
max tf_q	maximum term frequency of any term in Q
N	collection size

Sample Similarity Computation for ntc·atn Weights

Table 6

	Binary documents Binary Queries (Combination Level) bnn·bnn	Binary Documents Idf Queries (CM base case) bnn·btn	Improvement over Pure Binary Case
MED (1033 documents 30 queries)	0.4132	0.5098	+23%
CACM (3204 documents 52 queries	0.1848	0.2388	+29%
CISI (1460 documents 35 queries	0.1033	0.1226	+19%
INSPEC (12684 documents 77 queries	0.0944	0.1562	+65%

Importance of Inverse Document Frequency (idf) Weighting
Used in CM Base Case
(initial collection search)

Table 7

	Binary Docs Idf Queries (CM Base) bnn·btn	Normalized Term Frequency (tf) Docs Idf Queries ann·btn	Normalized tf·idf Documents and Idf Queries atn·btn
MED 1033	0.5098	0.5441 (+ 7%)	0.5502 (+ 8%)
CACM 3204	0.2388	0.3101 (+30%)	0.3095 (+30%)
CISI 1460	0.1226	0.1410 (+15%)	0.1528 (+25%)
INSPEC 12684	0.1562	0.2070 (+33%)	0.2236 (+43%)

Importance of Term Weighting for Documents
(with Conventional CM Queries)
(initial collection search)

Table 8

	Binary Docs Idf Queries (CM Base) bnn·btn	Tf × Idf Length Normalized Documents Idf Queries ntc·btn	Tf × Idf Length Normalized Documents Tf × Idf Queries ntc·atn
MED 1033	0.5098	0.5586 (+10%)	0.5628 (+11%)
CACM 3204	0.2388	0.3523 (+48%)	0.3630 (+52%)
CISI 1460	0.1226	0.1864 (+52%)	0.2189 (+79%)
INSPEC 12684	0.1562	0.2433 (+56%)	0.2626 (+68%)

Importance of Fancy Term Weighting for Documents and Queries
(initial collection search)

Table 9

	Initial Run Binary Docs Idf Queries (CM Base Case) bnn·btn	CM Relevance Feedback Run Binary Docs Idf Queries bnn·btn	Deterimination of First Feedback Run over Initial Run
MED 1033	0.2812	0.1438	-49%
CACM 3204	0.1283	0.0850	-34%
CISI 1460	0.0962	0.0644	-33%
INSPEC 12684	0.0905	0.0323	-64%

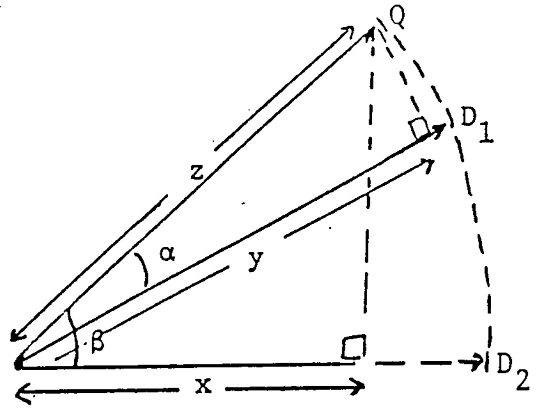
Evaluation of Connection Machine Relevance Feedback Method

Table 10

		All Relevance Feedback Runs Based on CM Base Run		
	Initial Run Binary Docs Idf Queries (CM Base) bnn·btn	Tf·Idf Length Normalized Docs Idf Queries ntc·btn	Tf·Idf Documents Weighted Query Adjustment (5) atn·IDE(ntc)	Tf·Idf Length Normalized Docs. Weighted Query Adjustment (5) ntc·IDE(ntc)
MED 1033	0.2812	0.4486 (+60%)	0.6425 (+128%)	0.8385 (+127%)
CACM 3204	0.1283	0.2035 (+59%)	0.2972 (+132%)	0.3301 (+157%)
CISI 1460	0.0962	0.1314 (+37%)	0.2101 (+118%)	0.2590 (+169%)
INSPEC 12684	0.0905	0.0884 (- 2%)	0.2282 (+152%)	0.2478 (+174%)

Importance of Weighted Documents and of Query Weight Adjustment
in Relevance Feedback

Table 11

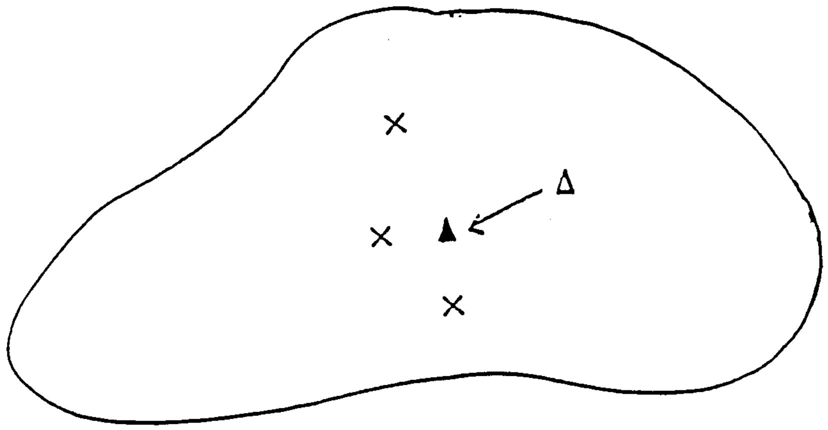


Vector Similarity Measurements

Fig. 1

$$\mathrm{Sim}(Q,D_1) = \cos\alpha = y/z; \quad \mathrm{Sim}(Q,D_2) = \cos\beta = x/z$$

$$y = Q\cos\alpha > x = Q\cos\beta$$



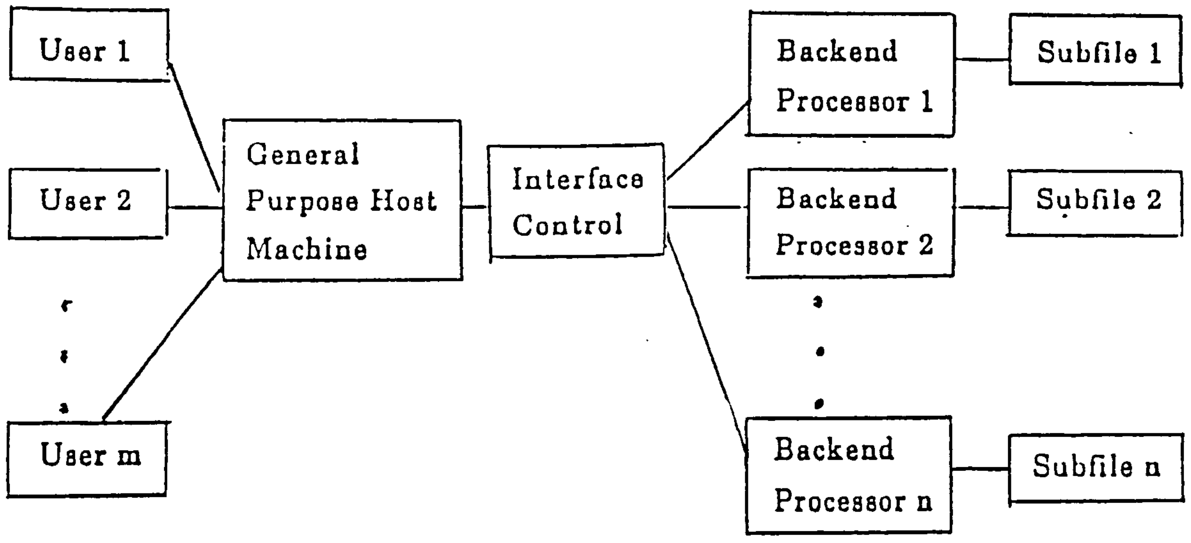
Δ original query

× retrieved items
designated as relevant

▲ altered query

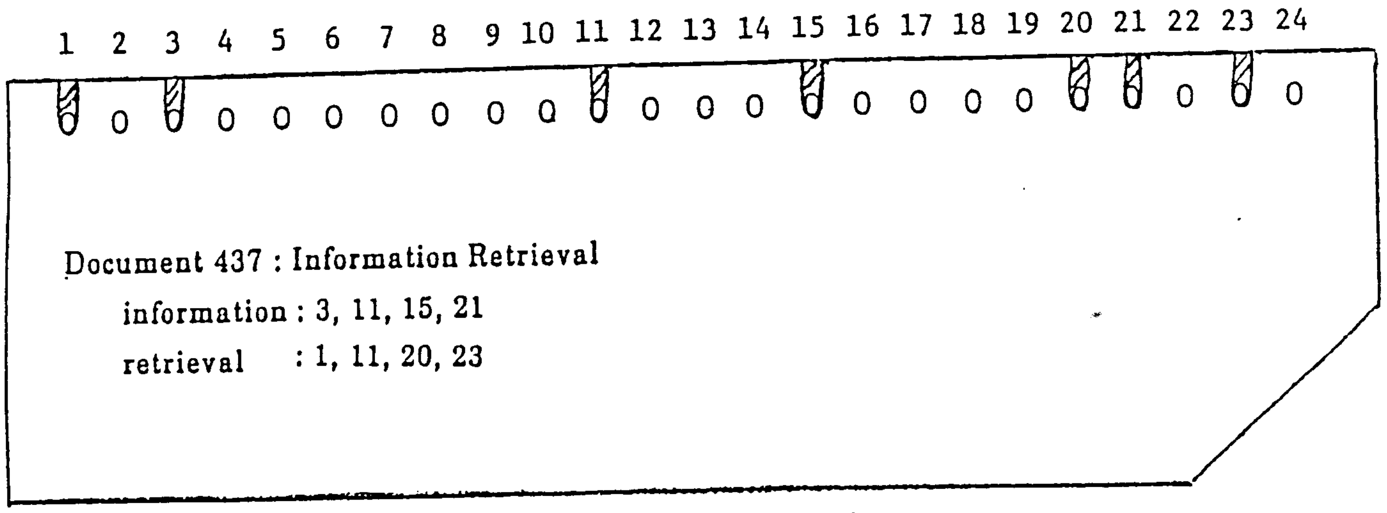
Relevance Feedback Illustration

Fig. 2



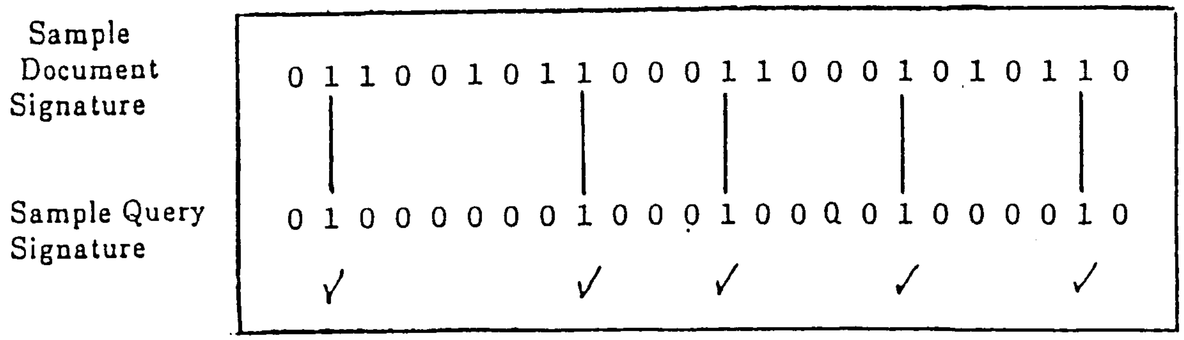
Configuration with Multiple Backend Search Processors

Fig. 3


1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
Document 437 : Information Retrieval
information : 3, 11, 15, 21
retrieval : 1, 11, 20, 23

Sample Edge Notched Card

Fig. 4



Sample Bit String Comparison