Optimization of Inverted Vector Searches

Chris Buckley
Department of Computer Science
Cornell University
Ithaca, New York 14853

Alan F. Lewit
College of the Virgin Islands
Box 84, Kingshill
St. Croix, Virgin Islands 00850

A simple algorithm is presented for increasing the efficiency of information retrieval searches which are implemented using inverted files. This optimization algorithm employs knowledge about the methods used for weighting document and query terms in order to examine as few inverted lists as possible. An extension to the basic algorithm allows greatly increased performance optimization at a modest cost in retrieval effectiveness. Experimental runs are made examining several different term weighting models and showing the optimization possible with each.

1. Introduction

Efficiently comparing a query to a collection of documents is a critical topic for information retrieval research. The vast store of information contained in any collection is useless if users are not willing to wait for the system to respond to their query. Not only is efficient comparison important for actual retrieval of documents, but many subsidiary activities, such as Maximum Spanning Tree generation or nearest neighbor searches, depend on it.

For the purposes of this paper, a query is considered to be a vector in n-dimensional space, each dimension corresponding to a concept which could potentially occur in the query. In particular, this paper does not look at Boolean queries. Each document is a vector in the same space; the similarity between the query and a document is a measure of how close the document vector is to the query vector.

The straight-forward method of finding the closest documents to a query is simply to go through the document collection sequentially. Each document vector is compared in turn to the query vector. This sequential search is fine for very small collections, but it is infeasible for most collections since so many similarities must be computed. Something must be done to cut down on the number of similarity computations.

One approach is to group the documents in a collection such that similar documents are clustered together.[1,2,3] Only those documents whose cluster representative has a high similarity to the query are candidates for retrieval. Unfortunately, no clustering method has yet proven itself to be effective across a wide variety of document collections.

Another approach, that of Eastman and Weiss[4,5,6], finds the nearest neighbor using a binary search tree with terms as node values. Any document appearing in the left sub-tree of a node has no terms in common with the node, a document in the right sub-tree has a least one term in common. At any point in the tree search, an upper bound can be placed on the similarity for all unseen documents. The search terminates when this upper bound becomes less than the similarity of a seen document. However, it is quite difficult to construct a good binary tree (possibly as hard as it is to cluster). In addition, only a small saving in efficiency was demonstrated on the small document collection used in these experiments.

A third approach comes from the observation that most documents in a collection have no terms in common with any given query. Therefore, an inverted file giving the documents in which a query term occurs can be used [7,8]. Query-document similarities will only be computed for

This study was supported in part by the National Science Foundation under grant IST 83-16166.

those documents which occur in an inverted list of some query term. In this way, each similarity is known to be non-zero since there will be at least one term in common. To further speed up retrieval, stopping conditions can be calculated so that not all of the inverted lists need to be examined. These stopping conditions consider the maximum similarity that the query can have with documents not yet examined. The maximum similarity is based on the particular retrieval method used, and on the minimum length of the documents remaining.

The problem with these methods for retrieval is that they were designed for another purpose, nearest neighbor matching of binary weighted vectors. They extend to the task of weighted retrieval in a straight-forward manner, but not efficiently. They also require a large number of random accesses to a direct document vector file in addition to the accesses to an inverted file. If the similarity function can be chosen such that all the necessary information can be stored within the inverted file, then the accesses to the document vector file can be eliminated.

The current paper examines one such similarity function, looking at several term weighting methods. Different stopping conditions are obtained for each weighting method. An inner product function is used. The similarity between a query and a document is simply the sum of the products of corresponding term weights. Note that this is not as restrictive a choice as it might appear at first glance. With appropriate term weighting functions, the inner product is equivalent to a number of other similarity functions. For example, the cosine function is identical to an inner product function with each document weight normalized by the length of the document it occurs in. An inverted file is used to store the list of documents and document weights associated with each term in the collection. The inverted file can be used to compute an inner product similarity function easily, in the manner of the SIRE system of McGill et al[9,10], and the CITE system of Doszkocs and Rapp[11].

This paper considers an approach in which query-document similarities are only partially calculated. The inverted file is used to access lists of documents corresponding to each term in the query. Each inverted list is accessed and a partial similarity due to the current term is computed and stored. As soon as these partial similarities determine the set of documents to be returned to the user, the retrieval operation is over, despite the fact that the total similarity of each of the returned documents may not have been computed yet. The stopping condition will be dependent on the term weights used.

This approach can be contrasted to that proposed by Harper[12] who also used an inverted file implementation of the inner product similarity function (although with binary document weights). Harper kept track of partial similarities for **only** those documents which might enter the set of documents to be returned to the user. His optimizations involved considering all inverted lists, but only calculating similarities for "possible" documents. The approach in this paper calculates partial similarities for all involved documents, but only considers the "useful" inverted lists. The two algorithms are actually complementary and with a bit of work could be merged. This work is left for the future, however.

The basic algorithm here can be extended for further operating efficiencies. Instead of stopping when the set of documents to be returned to the user is completely determined, retrieval can stop when the majority of the most similar documents are guaranteed to be among the retrieved set, but some highly similar documents may not be. There might be a slight degradation in retrieval effectiveness if this is done. Experimental results are presented that show the degradation is not serious and that the performance improvement is significant.

This approach is feasible now because of the comparatively large amounts of cheap memory available on today's machines. Ten years ago memory limitations would prohibit the storage of all of the partial similarities within main memory. The overhead associated with storing the partial similarities on disk would outweigh any advantage to be gained from using the partial similarities. Now, this method would be feasible for all but the largest collections (greater than 2 million documents).

2. The Basic Algorithm

Assume that the user is initially interested in looking at the top K documents for a particular query. Informally, the algorithm is as follows:

Consider the list of query terms in order of decreasing query term weight.

For each <document id, document term weight> pair in a query term's inverted list, add the partial similarity due to the present term to the partial similarity for this document already computed from previous terms.. Keep track of the top $K+1$ partial similarities computed so far.

Stop when enough terms have been looked at so that it is impossible for the $K+1^{st}$ ranked document to increase in rank, even if all of the remaining query terms occur in the $K+1^{st}$ ranked document.

The exact stopping condition used is really the only interesting part of the algorithm. As seen below, it depends greatly on the particular document and query weighting schemes used.

More formally, represent query Q by $<q_1, q_2, q_3, \cdots, q_t>$. Let I_{q_i} be the inverted list for term q_i with elements $d_{j,i}$ giving the weight of term i in document D_j.

```
Sort the q_i in order of decreasing query weight;
Set all elements of array SIM[1..N] to 0. (N is
    the number of documents in the collection).
    SIM[j] is the partial similarity computed thus
    far between query Q and document D_j;
Set num_top to 0;
Set all elements of array TOP_DOC[1..K+1] to 0.
    TOP_DOC[1..num_top] keeps, in decreasing
    similarity, the documents with the highest
    partial similarities computed.

foreach q_i in Q
   foreach d_{j,i} in I_{q_i}
      Set SIM[j] = SIM[j] + q_i * d_{j,i};
      If (num_top ≤ K  or
          SIM[j] > SIM[TOP_DOC[num_top]]) then
          If (j not in TOP_DOC[1..K+1]) then
             If (num_top < K+1) then
                Set num_top = num_top + 1
             end_if;

             Set TOP_DOC[num_top] = j;
          end_if;

          Restore TOP_DOC to sorted order;
      end_if;
   end_foreach;

   If (num_top = K+1) then
      Set max_remaining_sim to maximum possible
          partial similarity of document K+1 to
          the query terms not yet considered;

      If (SIM[TOP_DOC[K+1]]+ max_remaining_sim
              ≤ SIM[TOP_DOC[K]]) then
          Done - return TOP_DOC[1..K]
              documents to the user.
      end_if;
   end_if;
end_foreach;
```

It is now time to look at some examples of weighting schemes to see how *max_remaining_sim* can be bounded. Three common term weighting schemes are discussed below.

2.1. Normalized Document Vector

The weights in a document vector are often normalized so that a long document will not be retrieved before a short document just because of its length. Suppose the weights are normalized so that $\sum d_i = 1$. This can be done by taking an arbitrarily weighted vector and dividing each term weight by the sum of the term weights in the vector. Since the query terms are sorted by decreasing weight, the maximum partial similarity of a document to the query terms following q_i, the present query term, is bounded by $1 * q_{i+1}$. The maximum similarity will be achieved if a document consists of a single term matching q_{i+1}.

Actually, this can be improved on slightly because document $K+1$ already has a similarity due to matching one or more of the first i query terms. Therefore, the maximum document weight that can match query term q_{i+1} is:

1 - the document weight used in matching the first i query terms.

The worst case occurs if as little document weight as possible was used in calculating the existing partial similarity; i.e., the document matched the highest weighted query term. The first query term, q_1, is the maximum weight term in the query, so the minimum document weight is

$$\frac{SIM[TOP\_DOC[K+1]]}{q_1}$$

This yields the final upper bound for the remaining partial similarity after term q_i as

$$max\_remaining\_sim = q_{i+1}\left(1 - \frac{SIM[TOP\_DOC[K+1]]}{q_1}\right) \qquad \text{(E1)}$$

Note that this first stopping condition is entirely independent of the overall type of weighting scheme used; it depends only on the normalization of the document weight.

As an example, consider the situation where the user requests 5 documents with a query, Q, of

$$Q = (q_1{=}0.5,\ q_2{=}0.25,\ q_3{=}0.1,\ q_4{=}0.1,\ q_5{=}0.05)$$

Suppose the highest six partial similarity values after processing the first three query terms (q_1, q_2, q_3) are

Document_Id	Partial_Similarity
1	.45
2	.42
3	.40
4	.38
5	.35
6	.30

At this point, the algorithm should stop and return the top 5 documents to the user if it is impossible for document 6 to have a **final** similarity value of .35 or greater.

For document 6 to have its current partial similarity of .30, at least one of its terms had to match the first three query terms. The worst possible case is if document 6 matched query term q_1 with a document weight of 0.6. This maximizes the total weight which the rest of the terms in document 6 can have. Since the sum of the weights of the entire document is 1.0, the remaining terms of document 6 can have a maximum total weight of 0.4. If the entire weight of 0.4 matches the highest weighted query term remaining (term q_4), the maximum final similarity which could be achieved is .30 + .4 * .1 = .34. Therefore, it is impossible for document 6 to have a similarity greater than .35, and the retrieval algorithm should halt now.

2.2. Document Weight as a Function of Query Weight

Several weighting schemes compute both document weights and query weights in roughly the same fashion. A bound for the document weight can thus be computed from the query weight. An example of this is the *tf * idf* weighting function used by the SMART system for many years. Both the query and the document vectors are indexed with the same formula. Since the *idf* (*inverse document frequency*) component is dependent on features of the entire collection, the only difference between the weights in the two vectors is due to the difference in the term frequency component. The exact formula used in recent years by SMART is

$$weight = \left[0.5 + 0.5\,\frac{tf}{max\_tf}\right] \log\left(\frac{N}{n}\right) \qquad \text{(W1)}$$

where

tf = within document frequency of term
max_tf = maximum tf for all terms in vector
N = number of documents in the collection
n = number of documents which contain term.

The minimum value for the term frequency component is 0.5 and the maximum value is 1.0. Therefore, an upper-bound for the weight for a document term is twice the weight of that term in the query. This yields an equation for the remaining partial similarity after query term i of

$$max\_remaining\_sim = \sum_{j=i+1}^{t} q_j\,(2q_j) \qquad \text{(E2)}$$

A slight variation of this would be to normalize both the document and query vectors by the length of the respective vectors. If the document length is assumed to be greater than the query length, then E2 still gives an upper bound. This variation is equivalent to the classical cosine run using *tf*idf* weights.

2.3. Bounded Document Weight

A third type of weighting scheme is that in which there is a constant bound, say B, (normally 1) on the weight of a document term. Examples of this are:

Binary document weight

$$weight = 1 \qquad \text{(W2)}$$

Croft's Probabilistic weight [13, 14]

$$weight = Prob(d_i = 1) \qquad \text{(W3)}$$

Normalized term frequency

$$weight = \frac{tf}{max\_tf} \qquad \text{(W4)}$$

Augmented normalized term frequency

$$weight = 0.5 + 0.5\,\frac{tf}{max\_tf} \qquad \text{(W5)}$$

A bounded document weight situation, with bound B, gives the following equation:

$$max\_remaining\_sim = B * \sum_{j=i+1}^{t} q_j \qquad \text{(E3)}$$

3. Further Optimizations of the Basic Algorithm

Two major extensions to the basic optimization algorithm suggest themselves. The first extension is based on equations E1, E2, and E3 all being worst case equations. It is very unlikely that any document will ever satisfy the worst case conditions. Modifications to the equations could be made so it would be **expected** that no document would exceed *max_remaining_sim*. Unfortunately, changes of this type are extremely sensitive to both the particular weighting scheme in use, and the particular document collection. In the absence of any general theory of what kind of changes would work, this further optimization is not pursued.

A more workable extension comes from the observed fact that the chance of the K^{th} ranked document being relevant is not that much greater than the chance of the $K+1^{st}$ ranked document being relevant. Effectiveness should not suffer greatly if the $K+1^{st}$ document were to be returned instead of the K^{th} document. Instead of guaranteeing the top K documents will be returned to the user, the retrieval system can guarantee the top n (for $n<K$) documents will be returned, along with another $K-n$ documents which will have "high" similarity.

This extension can be achieved simply by using the similarity of the n^{th} ranked document instead of the K^{th} in the stopping condition:

```
If (SIM[TOP_DOC[K+1]] + max_remaining_sim
                    ≤ SIM[TOP_DOC[n]]) then
   Done - return TOP_DOC[1..K]
      documents to the user.
end_if;
```

The new stopping condition assures that every document with a total similarity greater than the current n^{th} ranked document's partial similarity is returned to the user. Therefore, at least the top n documents with highest total similarity will be returned.

4. Analysis of Algorithms

The major requirement for both the basic algorithm and the extended algorithm is that the document collection must be available in inverted list form with document term weights given in the inverted lists. There is no need for a sequential form of the document collection to be kept, where the individual document vectors are directly available. Several of the previous algorithms already mentioned required both the inverted list and sequential forms of the collection to be kept, eg. [7,8].

Each algorithm calls for the partial similarities to be kept within main memory. The current implementation uses 4 bytes per partial similarity and allocates enough memory to hold a partial similarity for every document in the collection. A collection with 100,000 documents would need 400,000 bytes of memory to store the partial similarities. The cost of a computer with sufficient memory is currently much less than the cost of a reasonably fast disk holding the documents and associated inverted lists.

As was previously mentioned, the extended algorithm is expected to achieve its performance improvement at a cost in retrieval effectiveness. Some highly similar documents may not be found by the time the extended algorithm satisfies its stopping conditions. The size of the effectiveness degradation will depend (among other things) on how closely the number of top documents actually retrieved matches the number of top documents guaranteed to be retrieved. For example, if the stopping condition guarantees that the top 3 documents will be among 10 documents returned to the user, but in practice the top 9 documents are retrieved, then there will be little change in effectiveness between the optimized and basic runs.

The documents returned to the user by either algorithm are not as strongly ranked as they would be without using any optimization. Even if the top 10 documents are returned to the user, they will be ranked in order of their partial similarity instead of total similarity. A user interested in high precision among the documents retrieved may not wish to use any optimization.

The running time of the algorithms is determined by the number of entries in the inverted lists for the query terms examined. The fewer the number of lists looked at, and the shorter these lists, the better the running time will be. Note that a term which occurs frequently in a collection is probably a poor term [15] and should receive a low query weight. Terms with long inverted lists will therefore be examined last since the query terms are sorted by decreasing query weight. The most expensive lists to look at will be the ones dropped because of optimization.

As presented, the algorithm does have a component with a running time linear in the number

of documents in the collection. This is the initialization of the array of partial similarities to 0. A clever implementation can avoid having to do this initialization explicitly, so this is not a problem.

5. Experimental Runs

Experimental runs exploring both the basic and the extended algorithm were done on two collections. The CACM document collection consists of the titles, abstracts, and other information for 3,204 computer science articles published in the *Communications of the ACM* between 1958 and 1979 [16]. The INSPEC collection contains the abstracts of 12,684 scientific documents. It was obtained from Syracuse University, with additional relevance assessments being done at Cornell. See table 1 for statistics on the two document collections. The experiments were performed using the SMART information retrieval system on a DEC VAX 11/780 under the UNIX operating system [17].

In order to examine the effect of query length on optimization performance, two sets of queries were used for each collection. The first set was the set of original natural language queries obtained from users. The *tf * idf* functions given in equations W1 and W5 were used to index this set. The other set was a set of longer feedback queries. These were obtained by retrieving the best 10 documents for each query in the original set. The original query terms were combined with terms from the corresponding relevant documents in order to form a new query. These query terms were weighted using a term-relevance weighting scheme [18]. Table 2 gives statistics for the four sets of queries.

Document weighting methods were chosen from each of the three categories discussed earlier. To test equation E1 (normalized document vectors), the document collections were indexed using weighting function W5. Each document was then normalized by dividing its term weights by the sum of the term weights in the vector. For equation E2, the documents were weighted using the same approach as the original queries (*tf * idf*) according to weighting function W1. Note that this approach could not be used with the feedback query sets, since the feedback query weights were assigned by the feedback method itself, and thus bore no relation to the document weights. The bounded document weight category (equation E3) was tested using weighting function W5 again, but for this category no normalization of the vectors was done.

For each combination of collection, query set, and document weighting method specified above, a set of 11 retrieval runs was made. Each run returned 10 documents to the user, but guaranteed only a given target number of the best documents in those 10. Target values 1 through 10 were all tested. A target value of 10 corresponded to the basic optimization procedure first presented. A target value less than 10 guaranteed that that many of the top documents were returned to the user. An unoptimized retrieval run was also performed.

5.1. Evaluation of Optimization

Tables 3-12 present the consequences of not examining all inverted lists corresponding to query terms. There are three main categories of effects given: the absolute savings due to the actual number of inverted lists looked at; the savings in CPU time and actual disk accesses caused by the lowered number of inverted lists; and the retrieval effectiveness which is changed because less information is available.

The "Num Con" sub-category under "Absolute Statistics" in each table gives the number of query concepts used (and thus the number of inverted lists). "Num Block" gives the estimated number of disk blocks that need to be read to get the inverted lists. Each list requires one disk access to learn the head of the list, and one or more disk accesses to obtain the actual inverted list, depending on the length of the list. The number of floating point multiplications done (of query weight times document weight) is given in the "Num Mult" field.

The sub-categories under "System Statistics" verify that the savings suggested by the absolute statistics are actually being realized by the retrieval system. "CPU Time" indicates the total amount of time spent by the computer on the retrieval run. This includes the work the operating system had to do to read data from the disk; it does not include the time spent waiting for the disk. The "Num Block" field gives the actual number of disk blocks read during the course of the retrieval. Since CACM is a "small" collection (3204 documents) and a number of inverted lists can be read with one disk block access, the actual number of disk blocks will be considerably smaller than the estimated number found under the "Absolute Statistics" category. *Note: the system statistics presented are load-sensitive. They vary depending on the other processes being run on the computer at the same time.*

Retrieval effectiveness is measured by the average recall at the 10 document cut-off level. Thus a recall figure of .3 means that, on average, 30% of the relevant documents for a query were retrieved within the top 10 documents. Note that for a feedback query, any document seen during the original query retrieval is no longer considered part of the collection. Therefore, .3 indicates that 30% of the *remaining* relevant documents were retrieved within the top 10.

5.2. Analysis of Results

We first examine several of the individual result tables. Table 3 gives the results for original query runs on CACM with document weights bounded by 1.0. The number of concepts examined does not fall quickly as the target decreases to 1. It is only when target reaches 2 that more than one concept is dropped per query on average. Despite this, the number of floating point multiplications done by this point has almost halved. Both query weighting methods (for original and feedback queries) will assign the lowest weights to terms which occur in many documents. The lowest weight query concepts are examined last, so the longest inverted lists will be the first ones dropped. Also, note the very small decrease in average recall as target approaches 1. There is a 3% recall degradation while there is a 37% CPU time improvement (comparing the base case and target = 1).

Table 5 gives the results for retrievals with normalized document vectors on the CACM collection. There are two features which are immediately obvious. The first is that very little optimization could be done. There wasn't enough information about the document weights to stop the retrieval process early. The other (unintended) feature is the poor recall values obtained. While the particular weighting scheme chosen is not one to be strongly recommended for any collection, it is particularly bad for the CACM collection. (For CACM, there is a strong correlation between length of a document and relevance. Documents published in early years tended to be short and non-relevant.)

The results of feedback query runs on the INSPEC collection are given in tables 10 and 12. Table 10 shows the results of optimization knowing that the document weight is bounded by 1. The performance improvement is tremendous! As target goes to 1, there is a 77% decrease in the estimated number of disk accesses, an 88% decrease in the number of floating point multiplications, and an 84% decrease in the CPU time spent. The recall penalty paid for this performance is beginning to be noticeable: 10%

In the runs shown in table 12, the document vector is known to be normalized. As opposed to table 5 (also using normalized document vectors), there is a substantial performance improvement. The absolute recall level is still low, but degrades very little as the efficiency greatly increases.

There are a number of general observations that can be made from Tables 3-12. Perhaps the most surprising one is that recall values do not degrade significantly as the number of guaranteed top documents goes from 10 towards 1. The worst effectiveness penalty is only 10% (in Table 10). This suggests that the most drastic optimizations given in this paper just barely reach the point where the result of optimization trades off retrieval effectiveness for retrieval efficiency.

As is to be expected, the more information that is known about the document weights involved, the more performance optimization can be done. The largest performance increase is consistently obtained when the document weight is known to be bounded by twice the query weight. The bounded document weight does almost as well in these experiments. Note that the particular weighting function used yields a weight which is very often close to the upper bound. If only the total length of the document vector is known, then the performance increase available is quite a bit less. There is still a significant performance improvement with a sufficiently long query.

For original queries, stopping the search immediately after all of the top 10 documents have been found does not prove worthwhile (this was the original basic optimization). There is no improvement in four original query runs, and only slight improvement in the other two. The longer feedback queries offer more opportunity for improvement; up to a 21% decrease in the number of disk blocks accessed is observed (see Table 10). Note that this rise in the efficiency level is obtained while guaranteeing no decrease in retrieval effectiveness.

The small deterioration in effectiveness as fewer query terms are considered suggests that some queries have a good number of redundant or even useless terms. The question of whether these terms can be eliminated during the query indexing process deserves further investigation.

Another contributing factor to the small change in effectiveness is that the upper bound for

max_remaining_sim is a **worst case** upper bound. It is rarely, if ever, achieved. A mathematical model of term weight distributions within a document will give a much smaller **expected** bound for *max_remaining_sim*. Use of a tighter bound would allow a user (or database administrator) greater options in trading off retrieval effectiveness for retrieval efficiency.

6. Conclusion

An algorithm has been presented which performs a retrieval using an inner product similarity function on vector collections with weighted terms. The algorithm uses an inverted file representation of the document collection and examines the minimum number of inverted lists possible in order to retrieve the best (most highly similar) documents. Fewer lists can be examined if the algorithm only assures that some of the best documents are retrieved. The optimization in the number of lists considered is based on knowledge about the particular term weighting method used.

Experimental results show that little performance improvement can be expected if all of the best documents are guaranteed to be retrieved. However, significant improvement (up to 88%) is achieved if the algorithm's guarantee is relaxed. Only a slight effectiveness deterioration results from this relaxation. This experimental evidence suggests that the extended algorithm is a practical optimization of the basic inverted file search.

References

[1] G. Salton, ed., *The SMART Retrieval System.* Prentice-Hall, Englewood Cliffs, N.J. (1971).

[2] R.E. Williamson, "Real-time Document Retrieval". Ph.D. Thesis, Cornell University (1971).

[3] N. Jardine and C.J. van Rijsbergen, "The Use of Hierarchic Clustering in Information Retrieval". *Inform. Stor. Retr.* 1971, **7**, 217-240.

[4] C.M. Eastman, "A Tree Based Algorithm for Nearest Neighbobr Searching in Document Retrieval Systems". Ph.D. Thesis, The University of North Carolina at Chapel Hill, (1977).

[5] C.M. Eastman and S.F. Weiss, "A Tree Algorithm for Nearest Neighbor Searching in Document Retrieval Systems". *Proc. Inter. Conference on Information Storage and Retrieval*, SIGIR, Rochester, N.Y. (1978)

[6] S.F. Weiss, "A Probabilistic Algorithm for Nearest Neighbor Searching". *Proc. ACM-BCS Symposium on Research and Development in IR*, Cambridge, England (1980).

[7] A.F. Smeaton and C.J. van Rijsbergen, "The Nearest Neighbour Problem in Information Retrieval". *Proc. Inter. Conference on Information Storage and Ret*, SIGIR, Oakland, California (1981).

[8] F. Murtagh, "A Very Fast Exact Nearest Neighbor Algorithm for Use in Information Retrieval". *Information Technology: Research and Development*, **1**, 1982, 275-283.

[9] M.J. McGill, T. Noreault, "Syracuse Information Retrieval Experiment (SIRE): Rationale and Basic System Design". Report, School of Information Studies, Syracue University, May 1977.

[10] M.J. McGill, L. Smith, S. Davidson, T. Noreault, "Syracuse Information Retrieval Experiment (SIRE): Design of an On-Line Bibliographic Retrieval System". *SIGIR Forum*, **10**, Spring 1976, 37-44.

[11] T.E. Doszkocs, B.A. Rapp, "Searching MEDLINE in English: A Prototype User Interface with Natural Language Query, Ranked Output and Relevance Feedback", *Proceedings of the ASIS Annual Meeting*, **16**, 1979, 131-139.

[12] D.J. Harper, "Relevance Feedback in Document Retrieval Systems: An Evaluation of Probabilistic Strategies". Ph.D. Thesis, The University of Cambridge (1980).

[13] W.B. Croft, "Document Representation in Probabilistic Models of Information Retrieval". *Journal of the American Society for Information Science*, **32**, 451-457.

[14] W.B. Croft, "Experiments with Representation in a Document Retrieval System". *Information Technology: Research and Development*, **2**, 1983, 1-21.

[15] G. Salton, C.S. Yang, C.T. Yu, "A Theory of Term Importance in Automatic Text Analysis". *Journal of the American Society for Information Science*, **26**, 1975, 33-44.

[16] E.A. Fox, "Characteristics of Two New Experimental Collections in Computer and Information Science Containing Textual and Bibliographic Concepts". Technical Report 83-561, Cornell University, 1983.

[17] C.A. Buckley, "An Overview of the Implementation of SMART". Technical Report, Cornell University, 1985.

[18] H. Wu and G. Salton, "The Estimation of Term Relevance Weights Using Relevance Feedback". *Journal of Documentation*, **37**, 1981, 194-214.

	CACM	INSPEC
Number of Documents	3204	12684
Number of Terms	17141	14573
Mean Terms per Doc	36.7	32.5

Table 1. Document Collection Characteristics

	CACM		INSPEC	
	Original	Feedback	Original	Feedback
Number of Queries	64	64	84	84
Number of Terms	378	1578	609	730
Mean Terms per Query	10.8	45.9	15.6	23.3
Mean Relevant Docs per Query	15.3	13.1	33.0	28.8

Table 2. Query Collection Characteristics

Target	Absolute Statistics			System Statistics		Recall
	Num Con	Num Block	Num Mult	CPU Time	Num Block	
1	589	1189	54217	9.6	545	.3001
2	623	1263	65688	10.9	583	.3173
3	643	1313	76073	12.0	611	.3122
4	655	1345	84049	12.8	625	.3120
5	669	1384	94448	13.3	646	.3092
6	678	1411	101750	14.3	654	.3130
7	680	1417	103304	14.3	655	.3130
8	686	1438	109706	15.4	729	.3130
9	687	1442	111022	15.4	708	.3130
10	689	1449	113118	15.5	712	.3120
Base	689	1449	113118	15.3	669	.3120

Table 3. CACM - Bounded Document Weight

Target	Absolute Statistics			System Statistics		Recall
	Num Con	Num Block	Num Mult	CPU Time	Num Block	
1	1132	2946	573153	61.3	2833	.1588
2	1178	3176	656545	68.2	3058	.1566
3	1213	3396	743683	75.9	3255	.1555
4	1241	3595	824413	83.1	3451	.1538
5	1265	3789	904855	90.3	3630	.1548
6	1276	3891	948261	93.6	3721	.1543
7	1291	4054	1019918	99.7	3863	.1501
8	1303	4191	1081440	104.5	3980	.1505
9	1309	4265	1114178	107.0	4033	.1508
10	1313	4317	1138152	109.2	4111	.1508
Base	1313	4317	1138152	109.7	4094	.1508

Table 4. INSPEC - Bounded Document Weight

Target	Absolute Statistics			System Statistics		Recall
	Num Con	Num Block	Num Mult	CPU Time	Num Block	
1	688	1445	111802	15.1	671	.0613
2	689	1449	113118	15.4	680	.0613
3	689	1449	113118	15.2	675	.0613
4	689	1449	113118	15.4	702	.0613
5	689	1449	113118	15.1	686	.0613
6	689	1449	113118	15.0	668	.0613
7	689	1449	113118	15.2	676	.0613
8	689	1449	113118	15.4	683	.0613
9	689	1449	113118	15.0	671	.0613
10	689	1449	113118	15.3	680	.0613
Base	689	1449	113118	15.4	699	.0613

Table 5. CACM - Normalized Document Vector

Target	Absolute Statistics			System Statistics		Recall
	Num Con	Num Block	Num Mult	CPU Time	Num Block	
1	1275	3957	982179	95.4	3797	.0567
2	1302	4194	1083213	105.6	4011	.0577
3	1312	4306	1133220	109.3	4084	.0577
4	1312	4306	1133220	108.6	4111	.0577
5	1313	4317	1138152	110.5	4184	.0577
6	1313	4317	1138152	111.4	4178	.0577
7	1313	4317	1138152	110.2	4105	.0577
8	1313	4317	1138152	109.3	4091	.0577
9	1313	4317	1138152	109.6	4096	.0577
10	1313	4317	1138152	110.9	4119	.0577
Base	1313	4317	1138152	111.8	4169	.0577

Table 6. INSPEC - Normalized Document Vector

Target	Absolute Statistics			System Statistics		Recall
	Num Con	Num Block	Num Mult	CPU Time	Num Block	
1	556	1116	43297	8.3	490	.2932
2	596	1199	53251	9.8	552	.2988
3	613	1235	58156	10.4	582	.3063
4	628	1271	66013	10.9	595	.3063
5	644	1312	75855	12.2	603	.3118
6	655	1343	83298	12.8	636	.3126
7	662	1364	88836	13.4	645	.3126
8	669	1384	93969	13.5	640	.3104
9	683	1426	105852	14.7	663	.3115
10	685	1434	108484	14.8	704	.3115
Base	689	1449	113118	15.3	675	.3115

Table 7. CACM - Document Weight as Function of Query Weight

Target	Absolute Statistics			System Statistics		Recall
	Num Con	Num Block	Num Mult	CPU Time	Num Block	
1	1000	2362	375294	44.8	2302	.1478
2	1082	2669	471754	53.4	2615	.1499
3	1123	2848	532421	58.6	2821	.1515
4	1157	3017	593231	65.8	3172	.1515
5	1178	3130	634603	69.1	3176	.1524
6	1205	3305	703269	75.4	3442	.1527
7	1233	3516	788301	81.3	3434	.1525
8	1253	3683	858989	86.0	3537	.1525
9	1278	3924	963865	95.4	3817	.1531
10	1297	4128	1053226	101.3	3847	.1561
Base	1313	4317	1138152	117.2	4927	.1561

Table 8. INSPEC - Document Weight as Function of Query Weight

Target	Absolute Statistics			System Statistics		Recall
	Num Con	Num Block	Num Mult	CPU Time	Num Block	
1	1467	2939	37113	16.4	1268	.2223
2	1637	3284	50810	18.5	1348	.2288
3	1785	3583	60191	20.8	1613	.2262
4	1919	3852	72041	21.7	1638	.2264
5	2052	4119	82324	23.4	1661	.2308
6	2193	4403	94928	25.7	1767	.2346
7	2333	4694	117529	28.5	1936	.2341
8	2465	4964	139080	31.9	2122	.2333
9	2577	5192	161292	34.3	2254	.2333
10	2775	5623	218899	40.0	2503	.2437
Base	2935	6000	289386	47.8	2754	.2437

Table 9. CACM - Feedback - Bounded Document Weight

Target	Absolute Statistics			System Statistics		Recall
	Num Con	Num Block	Num Mult	CPU Time	Num Block	
1	659	1544	230236	29.2	1390	.1211
2	761	1853	306433	36.4	1701	.1203
3	842	2101	367772	42.9	1946	.1259
4	978	2535	480119	53.6	2386	.1277
5	1084	2858	559626	63.5	2758	.1322
6	1195	3282	690624	74.2	3148	.1328
7	1308	3669	802142	82.9	3558	.1342
8	1434	4135	938614	96.4	3996	.1342
9	1538	4612	1100732	110.7	4465	.1355
10	1712	5427	1383291	136.2	5245	.1355
Base	1954	6876	1937397	185.3	6542	.1355

Table 10. INSPEC - Feedback - Bounded Document Weight

Target	Absolute Statistics			System Statistics		Recall
	Num Con	Num Block	Num Mult	CPU Time	Num Block	
1	1665	3347	62240	19.3	1387	.1588
2	2002	4023	88168	24.0	1645	.1676
3	2202	4427	110713	27.0	1896	.1668
4	2375	4779	131918	30.3	2046	.1622
5	2527	5098	155268	33.3	2142	.1668
6	2613	5280	176078	36.1	2393	.1668
7	2720	5507	204986	39.2	2674	.1668
8	2812	5712	237261	41.8	2562	.1668
9	2885	5877	263688	45.5	2865	.1668
10	2911	5940	276387	47.8	2862	.1668
Base	2935	6000	289386	49.5	2975	.1668

Table 11. CACM - Feedback - Normalized Document Vector

Target	Absolute Statistics			System Statistics		Recall
	Num Con	Num Block	Num Mult	CPU Time	Num Block	
1	675	1658	264651	32.0	1446	.0808
2	824	2099	370981	42.2	1869	.0816
3	953	2494	473254	51.7	2287	.0816
4	1097	2930	584009	62.1	2758	.0824
5	1206	3299	686639	73.3	3184	.0830
6	1315	3721	817201	84.8	3586	.0821
7	1422	4168	962177	102.5	4002	.0821
8	1554	4673	1119342	117.7	4542	.0821
9	1686	5397	1385397	140.7	5323	.0824
10	1809	6039	1620427	159.7	5777	.0824
Base	1954	6876	1937397	189.0	6665	.0824

Table 12. INSPEC - Feedback - Normalized Document Vector