A TREE ALGORITHM FOR NEAREST NEIGHBOR SEARCHING IN DOCUMENT RETRIEVAL SYSTEMS

By

Caroline M. Eastman
Department of Mathematics
Florida State University
Tallahassee, Florida 32306

Stephen F. Weiss
University of North Carolina
Chapel Hill, North Carolina

Abstract

The problem of finding nearest neighbors to a query in a document collection is a special case of associative retrieval, in which searches are performed using more than one key. A nearest neighbors associative retrieval algorithm, suitable for document retrieval using similarity matching, is described. The basic structure used is a binary tree, at each node a set of keys (concepts) is tested to select the most promising branch. Backtracking to initially rejected branches is allowed and often necessary.

Under certain conditions, the search time required by this algorithm is $O(\log_2 N)^k$. N is the number of documents, and k is a system-dependent parameter. A series of experiments with a small collection confirm the predictions made using the analytic model; k is approximately 4 in this situation.

This algorithm is compared with two other searching algorithms; sequential search and clustered search. For large collections, the average search time for this algorithm is less than that for a sequential search and greater than that for a clustered search. However, the clustered search,unlike the sequential search and this algorithm, does not guarantee that the near neighbors found are actually the nearest neighbors.

1. Introduction

This paper describes a searching algorithm that can be used in document retrieval systems and that appears to have some advantages over currently suggested methods. It could also be used in other similar searching problems.

The type of document retrieval system considered here is that described by Salton (1975) and used in the SMART system. In such a system, a set of concepts is used to classify both the documents and the queries in order to represent them as concept vectors. A similarity measure is used to compare documents and queries in concept vector form in order to select those documents best matching the queries; this approach may be contrasted with the use of Boolean matching.

The searches involved in such a system require the use of more than one search key; they are thus multi-attribute, or associative, searches. Since each concept is used as a key, the search space is of high dimension. Such a search is a nearest neighbor search; it looks for those documents which most closely match the query, according to any one of a variety of similarity measures (e.g. n-dimensional cosine, Euclidean distance). So it is a best-match search rather than an exact-match search or a partial-match search. Several documents are generally retrieved in response to a query; the search is thus an m nearest neighbor search rather than simply a nearest neighbor search.

The standard searching methods for document retrieval systems are sequential search and inverted file search. The sequential search is straightforward, but in large collections it is very time-consuming to examine the entire collection. The inverted file search is often used to provide quick searches in systems using Boolean matching; with somewhat more effort, it can be used in similarity-based searching as well. Inverted file searches become rather inefficient when requests have many concepts (Salton, 1968).

Clustered files are often suggested as a way to cut down search time in similarity-based systems. In such an organization, similar documents are grouped together in clusters, and only the most promising clusters are examined. Although such an approach can drastically reduce the search time, the near neighbors retrieved by a clustered file search are not always the nearest neighbors to the query. An overview of clustering methods is given in Salton (1975). Yu and Luk (1977) describe a model which provides an estimate of the number of nearest neighbors missed in a cluster search.

Rivest (1974) demonstrates the optimality of a particular hashing algorithm for a restricted class of best-match searches. However, he has assumed that the attributes are binary, that the records are randomly distributed, and that only one nearest neighbor is to be found; none of these conditions is met here.

Various tree-based methods have been suggested for retrieving m nearest neighbors. Quad trees (Finkle and Bentley, 1974) and k-d trees (Friedman, Bentley, and Finkle, 1977) are fine for small numbers of keys, but become impractical when lots of keys are involved. Burkhard and Keller (1973) and Fukanaga and Narendra (1975)

describe tree structured algorithms similar to the one suggested here; however, these two algorithms depend on the use of a distance metric rather than a similarity measure for matching.

2. Description

Each document in a collection of N documents is represented by a concept vector. Each query is also represented as a concept vector. The m documents most similar to the query are to be chosen. A similarity measure ranging between 0 and 1 is used to compare documents and queries. (A similarity of 1 means that the document and query are identical, and a similarity of 0 means that the document and the query contain no concepts in common.) The search algorithm described does not depend on the use of a particular similarity measure.

The document collection is organized as a binary tree. Associated with each internal node of the tree is a set of concepts. The documents are stored in buckets at the leaves of the tree; each document resides in exactly one bucket.

A document is inserted into the subtree rooted at a particular node as follows. The document is compared to the concept set associated with the node. If the intersection between the two is empty, then the document is inseted recursively in the tree whose root is the left child of the original node. If the intersection is non-empty, then the document is inserted in the right subtree. This intersect and descend process continues until a leaf node is reached; the document is then placed in the bucket associated with that leaf node. Thus, for each node, all documents that are in the right

subtree contain at least one concept in common with that node's concept set. All left-descendent documents contain none of the concepts.

The time required to insert one document into a tree of height L is $O(L)$. The maximum reasonable value of L for a collection of size N is $\log_2 N$. Such a tree would have an average bucket size of one. Any higher tree would guarantee empty buckets. Thus, in general, $L \leq \log_2 N$ and hence putting N documents into the tree is at worst $O(N \log_2 N)$.

The search is a similar process. The query is compared with the concept set of the root of a tree. If the intersection is empty then the left subtree is more likely to contain near neighbors of the query. If the query and the concept set have concepts in common, then the right subtree is more promising. In either case, the search descends to the appropriate subtree.

A bound can be put on the similarity between the query and the documents in a particular subtree. The nearest possible neighbor that could be found in a subtree can be determined by considering the concept sets to be found on the path to that subtree. This nearest neighbor need not actually be present in a particular tree; it is a theoretical bound. The similarity between the query and its nearest possible bound will be referred to as the similarity bound for that subtree. Of course, the calculation of the similarity bound will depend on the particular similarity measure used.

The similarity bound for the entire tree is 1 since the nearest possible neighbor is one which exactly matches the query. After

the query is compared to the concept set at the root, a similarity bound for both subtrees can be calculated.

Suppose the query and the concept set have c concepts in common. Then all documents in the left subtree are missing at least c of the concepts in the query. The nearest possible neighbor along the left path is thus a document that matches all but those c query concepts. The nearest possible neighbor for the right subtree is still an exact match to the query; the similarity bound for the subtree remains equal to 1.

If the query and the concept set have no concepts in common, all documents in the right subtree have at least one concept that was not requested. The nearest possible neighbor in the right subtree is thus a document that has all the query concepts plus one extra. The nearest possible neighbor for the left subtree is an exact match to the query.

Whether or not the query matches any of the concepts at the root node, the similarity bound for the less promising subtree can be calculated by determining the similarity between the query and its nearest possible neighbor in that subtree. The similarity bound for the more promising subtree remains 1.

Similarity bounds for subtrees of nodes below the root can be determined in much the same way. When a concept set is matched, the nearest possible neighbor and similarity bound remain the same for the right subtree. The matched concepts are not present in documents in the left subtree. So the nearest possible neighbor for the left subtree is the previous nearest possible neighbor

without the matched concepts. The similarity bound for this subtree is the similarity of the query with this nearest possible neighbor.

When a concept set is not matched, the nearest possible neighbor and similarity bound for the left subtree remain the same.

An extra concept is added to the nearest possible neighbor for that tree to obtain the nearest possible neighbor for its right subtree. The similarity bound can then be determined from this nearest possible neighbor.

The search process descends the tree until a leaf is reached. All documents in the associated bucket are compared with the query. If m nearest neighbors have been requested, the best m are saved. As before, the process of descending from the root to a bucket requires O(L) time.

Backtracking now occurs. If the similarity bound of any subtree encountered is greater than the similarity with the mth nearest neighbor so far encountered, that subtree is examined. There may be documents associated with it which are better than those already found. The search is over when the root of the tree is reached.

The search algorithm is summarized in Figure 1. The search is set in motion by calling SEARCH(Query, number of nearest nieghbors, root of entire tree).

Figure 1: Search Algorithm

```
SEARCH(QUERY,M,ROOT)
    IF (the ROOT is a leaf) THEN
        Compare each document in the associated bucket with the
        QUERY.  Merge the results with the  m  best documents
        seen so far; keep the best  m .
    IF (ROOT concept set  ∩ QUERY ≠ ∅) THEN
        Calculate the similarity bound for the left branch.
        CALL SEARCH(right child of ROOT).
        IF (the similarity bound for the left child is greater
            than the similarity of the  mth  best document found
            so far) THEN CALL SEARCH(left child of ROOT).
    IF (ROOT concept set  ∩ QUERY = ∅) THEN
        Calculate the similarity bound for the right branch.
        CALL SEARCH(left child of ROOT).
        IF (the similarity bound for the right child is greater
            than the similarity of the  mth  best document found
            so far) THEN CALL SEARCH(right child of ROOT).
END SEARCH
```

3. Analysis

If some restrictive but reasonable assumptions are made, the average time complexity of this algorithm can be determined. These assumptions are:

(1) The document collection is organized into a full binary tree of height L.

(2) Each bucket contains $\frac{N}{2^L}$ documents.

(3) The cutoff similarity, s, for the m nearest neighbors is the same for all queries. In other words, the m nearest neighbors to a query have similarities greater than or equal to s; all other documents have similarities less than or equal to s. (In a real situation, the value of s would almost certainly not be the same for all queries.)

(4) At any node, the reduction in the similarity bound for the subtree not initially chosen is r. (This assumption is clearly only a rough approximation to the actual situation.)

(5) Backtracking through the tree occurs in similarity bound order. That is, if the subtrees yet to be examined have similarity bounds, $s_1, s_2, \ldots s_j$, then the subtree examined nest is the one with bound equal to $MAX(s_1, s_2, , , s_j)$.

If these assumptions are made, then the number of buckets to be examined in a search is $\sum_{i=1}^{k} \binom{L}{i}$, where k is equal to $\lfloor \frac{1-s}{r} \rfloor$. If k is less than $L/2$, then this quantity can be shown to be $O(\log_2 N)^k$ and may be taken as the average time complexity.

(Details of this derivation are given in Eastman (1977).) Clearly this algorithm will be of practical interest only when k is small.

Large values of s (near 1) will result in small values of k. Such large values of s mean simply that the nearest neighbors to be found are very similar to the query. It should be noted that the value of s depends in no way on the tree.

Large values of r (near 1) will also result in small values of k. Large values of r mean that the similarity bounds for the branches not chosen are rapidly reduced. It is quite reasonable that a rapid tightening of the bounds would result in shorter searches by eliminating entire subtrees from further consideration. The value of r, unlike that of s, does depend on the particular tree used.

The behavior of this algorithm is intermediate between that of an $O(N)$ algorithm and that of an $O(\log_2 N)$ algorithm. For any given k, N grows faster than $(\log_2 N)^k$ as N approaches ∞ If k is large, however, the values of N for which $(\log_2 N)^k$ is smaller than N may be so enormous that they are much larger than any that would arise in a practical situation.

Figure 2 shows a comparison of N and $(\log_2 N)^4$. (The value of $k = 4$ was used because that was the value found in the experimental situation described in the next section). This graph illustrates the general shape of the two curves.

A more precise estimate of the expected search length can be obtained by dividing the number of buckets to be searched, $\sum_{i=1}^{k} \binom{L}{i}$, by the total number of buckets, 2^L. Figure 3 shows the predicted fraction of documents examined for several values of N. (These figures do not include time spent searching through the tree; this

time should be roughly proportional to the scan fraction). Although the algorithm performs poorly in small collections, it should do well in large ones.

Figure 2: Growth of N and $(\log_2 N)^4$

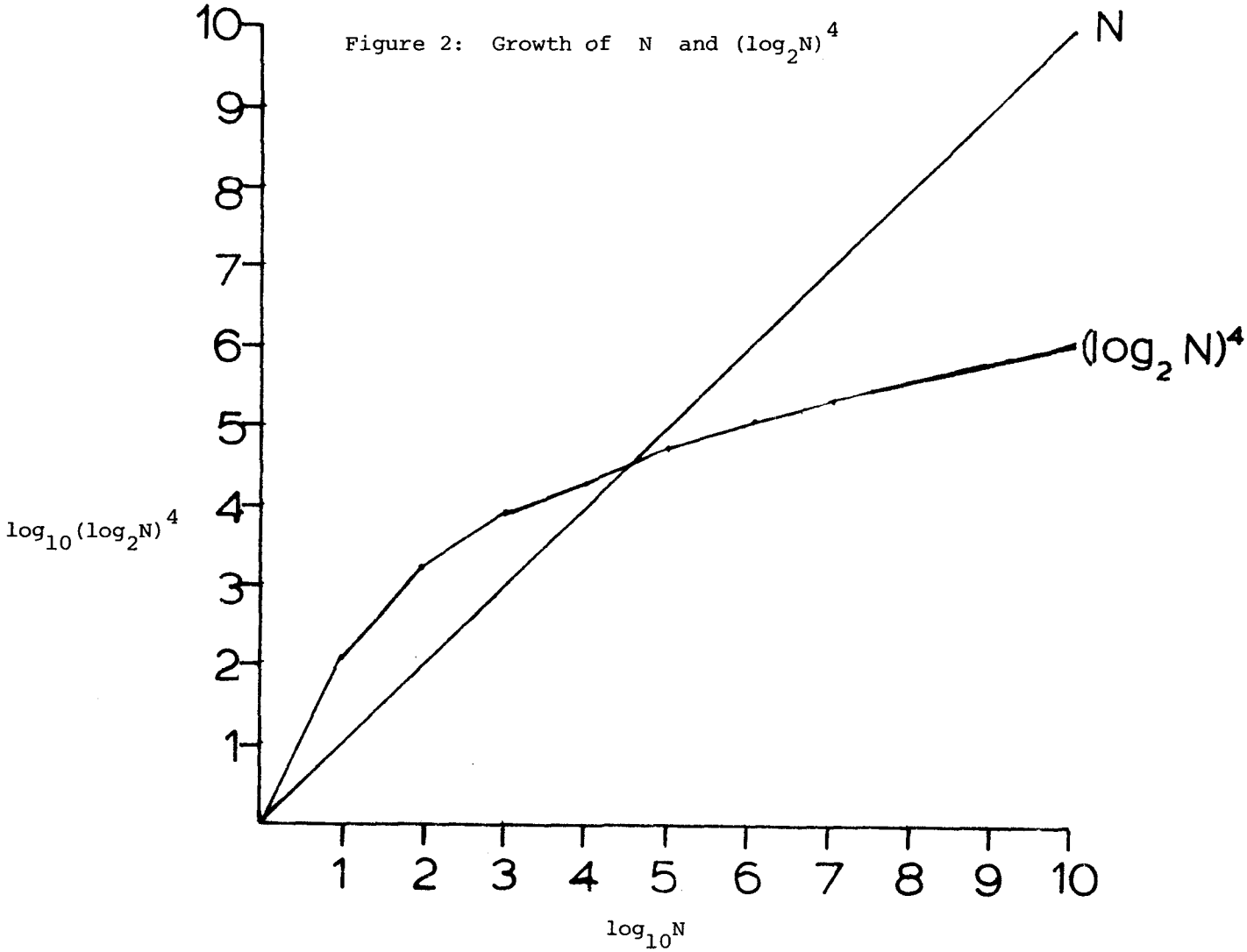

Figure 3: Some predicted scan fractions

(bucket size = 1)

	Scan Fraction	
N	k = 3	k = 5
128	0.50	0.94
1,024	0.17	0.72
1,048,576	0.0013	0.021

4. Experimental Results

Several experiments using this search algorithm were performed. These experiments were directed towards three related questions. First, how should the concept sets be chosen? Second, how useful is the rough analysis described in the previous section? Third, what value of k would be found in the actual situation?

The experiments described here used the American Documentation Institute (ADI) collection (Keen, 1971). The ADI collection contains 83 documents and 35 queries in concept vector form. It has been used in many document retrieval experiments based on the concept vector model of document retrieval.

As is generally true when there are a large number of factors influencing an outcome, it is infeasible to try all combinations of possible values for these variables. So a reasonable (but arbitrary) set of base conditions was chosen. These conditions are described in the next paragraph.

The 5 nearest neighbors are retrieved for each of the queries; this number was chosen because there are an average of 5 relevant documents per query for this collection. (In general, the fraction of the collection that is retrieved will be much smaller than the 6.1% retrieved here.) The concept tree used is a full binary tree of height 6; this height gives the average bucket size closest to 1 for a collection of 82 documents. Concept sets at the same level in a tree are identical; with a small number of concepts, it is easier to allow some duplication of concept sets. The nodes are searched in similarity bound order. The similarity measure used is the cosine.

The experimental conditions approximated the analytic assumption

described in the previous section. The tree used was a full binary tree, but the number of documents per bucket varied. The subtrees were searched in similarity ceiling order. The average value of s, the cutoff similarity for 5 documents, was 0.34. The reduction in similarity bound, r, varied with the direction chosen and the level of the node.

The measure used to compare the different searches is the average scan fraction, the fraction of documents searched in response to a query. Although the total work involved, including the tree traversal, is important, it should be roughly proportional to the scan fraction for trees of equal height.

Even though it is not clear what the optimal tree for a particular collection is, it is obvious that the selection of concept sets is critical to the performance of this algorithm. A greater chance of a match between a query and a concept set should increase the average r and thus decrease the average search time. Unfortunately, more sophisticated choice methods designed to increase the probability of such matches involve correspondingly greater effort.

Several methods of increasing sohpistication and overhead were used to construct concept trees. For each method, trees with concept sets of different sizes were constructed; the size giving the best performance for each method was used in the final comparison. In each case, the concepts used in the ADI queries were used as a source. (The same queries were used for constructing the tree and for searching. Of course, a production implementation would have to use some sample of queries.)

The simplest method of concept choice is random selection. A

somewhat more sophisticated method uses concept frequencies; the concept that occur most often in the sample of queries are chosen to form the concept sets. Since the most common concepts are used, there should be more matches.

Use of concepts highly correlated with each other should increase the probability of multiple matches with concept sets and decrease the search length. Two possible algorithms using correlations were used. The first forms the sets one at a time by taking the most common unused concept to start a set and then selecting those unused concepts most highly correlated with it. By adding one element to each set in turn, the second selection algorithm avoids the possibility that the very highest frequency concepts will all be in the same set.

A more sophisticated way of using the correlation information is to cluster the concepts and then choose concept sets from those clusters. Three of the many possible algorithm using clustering were tried. Two cluster the concepts using the single-link criterion until a cutoff similarity is reached. One algorithm, referred to as frequency-ordered clustering, then selected those clusters containing the most common concepts for concept sets. A second algorithm,referred to as size-ordered clustering, selected those clusters containing the most concepts.

The third clustering algorithm uses the most common concepts as seeds to start the clusters. If any two of the potential seed concepts are highly correlated with each other, they are put in the same cluster. Then single-link clustering is used to bring the clusters up to the desired size.

The results from searches using each of these methods are shown below in Figure 4.

Figure 4: Scan Fraction Using Different Methods of Concept Choise

Method	Scan Fraction	Relevant Parameter
Random	0.97	8 concepts/set
Most Common	0.92	4 concepts/set
Correlated (version 1)	0.89	8 concepts/set
Correlated (version 2)	0.89	8 concepts/set
Size-Ordered Clustered	0.94	Cutoff similarity = 0.55
Frequency-Ordered Clustered	0.94	Cutoff similarity = 0.55
Most Common Clustered	0.89	8 concepts/set

Figure 5: Increase In Scan Fraction with Number of Nearest Neighbors Retrieved

m	Average s	Scan Fraction
1	0.52	0.80
2	0.43	0.83
3	0.39	0.86
4	0.37	0.87
5	0.34	0.89
10	0.27	0.93
30	0.14	0.94
82	0.00	0.94

A Friedman analysis of variance was performed to test the null hypothesis that the concept selection method does not effect the scan fraction. The 35 scan fractions (one for each query) for the 7 different methods was used. The test statistic (χ_r^2) calculated is 20.7; the level of significance for this value is 0.002. So it is reasonable to conclude that the various methods of concept set formation differ significantly. Probably any method which uses the most common concepts to start the sets and then fills them out with additional concepts highly correlated with those already chosen would be a reasonable choice in practice. The three methods taking this approach had the lowest best scan fraction, 0.89 in each case.

The results from this experiment are in accord with the prediction, based on the analytic model, that those methods incorporating information about frequencies and clustering would give the best results. These methods should decrease the value of k by increasing the value of r.

The influence of the value of s upon the search length can be seen by looking at the scan fraction when different numbers of nearest neighbors are retrieved. A series of searches, retrieving different numbers of nearest neighbors, was performed using the Most Common Clustered tree with 8 concepts per set. The results are shown in Figure 5. The number of nearest neighbors actually retrieved was less than m if there were fewer than m documents with similarity greater than 0 to the query.

The scan fraction rises sharply from 0.80, the minimum obtained when 1 nearest neighbor is retrieved, to a plateau of 0.94. When m is 10, the average scan fraction is 0.93; this value is close to the plateau, which is reached when m is 30. The maximum

scan fraction is less than 1 because documents in subtrees with similarity bounds of 0 need not be searched.

As predicted, high scan fractions occur with relatively few documents retrieved in this collection. However, the behavior of the algorithm is such that much better results should be observed in large collections.

The result of the two experiments described here are in accord with predictions made on the basis of the analytic model. Several other experiments are described in Eastman (1977); the results of these experiments also agree with predictions made on the basis on the analysis. So it would appear that even the rough analysis described here can be useful in examining the behavior of this algorithm.

A rough value of k for the ADI collection can be estimated in two different ways. First, a value of k can be estimated from the observed scan fractions. Second, an estimate of k can be based on the estimated value of s and r. Both methods give an estimate of 4.

The expected scan fractions for a tree of height 6 can be obtained by dividing the estimated number of buckets searched, $\sum_{i=1}^{k} \binom{L}{i}$, by the number of buckets, 2^L. The lowest scan fraction obtained when 4 nearest neighbors were found in the ADI collection is 0.89. This is the scan fraction predicted when k is 4.

The average value of s when 5 neighbors are retrieved in the ADI collection is 0.52. The average query length of an ADI query is 4. The reductions in similarity bound occurring for 1 to 4 unmatched concepts and for 1 to 6 extra concepts for a query of

this length were calculated. The average r when matches occur is 0.25. The average r when matches do not occur is 0.06. Since most searches will involve a mixture of matches and misses, these two values were averaged to obtain an estimate for r of 0.16. Since k is equal to $\lfloor\frac{1-s}{r}\rfloor$, the estimate of k for this method is also 4.

5. Conclusions

For similarity-based document retrieval systems, this algorithm has advantages over both sequential search and clustered search. It offers a middle ground in terms of retrieval quality and search time.

This algorithm retrieves the same items as a sequential search, but the average search length is $O(\log_2 N)^k$ rather than $O(N)$. If k is approximately 4 in a large collection, as it was in the ADI collection, only a small fraction of the entire collection would need to be searched.

The algorithm described here requires more search time than a clustered search, both for the average case and for the worst case. However, it does a better job of retrieval. The items retrieved are actually the nearest neighbors rather than near neighbors to the query. Also, as the document collection grows, the retrieval degradation for a clustered search would occur in retrieval quality rather than search length. The tree-structured algorithm would still retrieve the nearest neighbors, so the speed, rather than retrieval quality, would suffer.

References

Burkhard, W.A. and Keller, R.M., 1973. "Some approaches to best-match file searching". Communications of the ACM, Vol. 16, No. 4, April 1973, pp. 230-236.

Eastman, Caroline M. "A tree algorithm for nearest neighbor searching in document retrieval systems". Ph.D. Dissertation, University of North Carolina at Chapel Hill, 1977.

Finkel, R.A. and Bentley, J.L. "Quad trees: a data structure for retrieval on composite keys". Acta informatica, Vol. 4, No. 1. 1974, pp. 1-9.

Friedman, J.H., Bentley, J.L. and Finkel, R.A. "An algorithm for finding best matches in logarithmic expected time". ACM Transactions on Mathematical Software, Vol. 3, No. 3, Sept. 1977, pp. 209-226.

Fukanaga, Keinosoke and Narendra, Patrenahalli, 1975. "A branch and bound algorithm for computing k-nearest neighbors". IEEE Transactions on Computers, Vol. C-24, No. 7, July, 1975, pp. 750-753.

Keen, E.M. "An Analysis of the documentation requests". In Salton, 1971.

Rivest, R.L. Analysis of associative retrieval algorithms. Stanford Computer Science Department, Report STAN-CS-74-415, 1974.

Salton, G. Automatic Information Organization and Retrieval. Mc-Graw-Hill Book Company, New York, New York, 1968.

Salton, G. editor, The SMART Retrieval System: Experiments in Automatic Document Processing. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1971.

Salton, G. Dynamic Information and Library Processing. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1975.

Yu, C.T. and Lik, W.S. "Analysis of effectiveness of retrieval in clustered files". Journal of the ACM, Vol. 24, No. 4, Oct. 1977, pp. 607-622.