A nearest neighbour search algorithm for bibliographic retrieval from multilist files

Peter Willett

Nearest neighbour, or best match, retrieval involves the identification of the document or documents most similar to some query specification, the degree of similarity being based upon the numbers of terms common and not common to a document and the query. Several upperbound best match search algorithms have been described recently that permit the efficient identification of such documents using inverted files. The present paper extends this work to cover document collections that are based on the multilist file organization. A new nearest neighbour algorithm, based on Hsiao and Harary's parallel algorithm for generalized files, is presented and the search efficiency tested with several sets of documents and queries: it is found that the algorithm can lead to a marked reduction in computation over a conventional nearest neighbour search if the exhaustivity of the query indexing is not high. The results are compared with those obtainable from the use of an alternative algorithm that is derived from earlier work on searching inverted files.

INTRODUCTION

Smeaton and van Rijsbergen[1] define the nearest neighbour problem as 'Given a set of N points in n-space and a distinguished point, Q, find the m ($m < N$) points 'closest' to Q closeness being measured by some distance measure'. Typical applications of the nearest neighbour, or best match, problem in bibliographic databases include finding the documents most similar to a query for the purpose of document retrieval, and the documents, or terms, closest to some specified document, or term, in various classification procedures. Because of the importance of nearest neighbour searching, there have been several attempts to develop algorithms that ensure the identification of the best matches for Q without the need to inspect all of the other points[1–4].

Much of the work on nearest neighbour searching has involved the use of the inverted file organization since this is widely used for the implementation of present-day commercial bibliographic retrieval systems. Although permitting a rapid response to Boolean partial match queries, such files present severe problems if online updating is required to add new records to the file, or to modify or delete some of those already present. An alternative file organization, the multilist file structure, may be used where such updating facilities are required, although this flexibility is counterbalanced in part by the increased retrieval times that may be expected when compared with the inverted file[5–6]. Multilist files are currently encountered primarily in commercial data processing environments, but it is to be expected that they will be increasingly used with textual data in the volatile files of letters, memoranda, and reports that are becoming available with the spread of integrated office automation systems.

After a brief description of the multilist file organization, this paper reports a new nearest neighbour algorithm for use with such files, and tests the search efficiency of the algorithm on a range of document and query sets.

Department of Information Studies, University of Sheffield, Western Bank, Sheffield S10 2TN, UK
Received 5 January 1984, revised 20 January 1984

 0144–817X/84/020078–06$03.00

MULTILIST PROCESSING

Introduction

A multilist file for a document collection consists of a data file, which contains the individual document records, together with an associated directory. Each entry in this directory corresponds to one of the keywords or indexing terms that have been used for the indexing of the document collection, and besides containing various data about that term, such as the collection frequency and the field type, also contains a pointer, i.e. an identifier or a relative or absolute disc address, to the *first* document in the collection to which that term has been assigned. The data records are augmented by a series of pointers, one for each of the terms in a document, that indicate the *next* document containing the corresponding term: thus a document containing n terms will have an associated set of n pointers and access to subsequent documents possessing these terms may be obtained only *via* this pointer set. If there are no further documents containing some term, the corresponding pointer is set to some null value. A retrieval operation will hence be initiated by access to the directory and continued by repeated reference to the data file; in an inverted file structure, conversely, the data file is required only for the presentation to the user of those records that satisfy the constraints imposed by the query.

Hsiao and Harary[8] have described a generalized file structure, of which the multilist is a special case, and proposed two retrieval algorithms for use with such structures. The first of these involves the processing of all of the records on one keyword list before proceeding to the records on the next, while the second search is carried out using all of the keyword lists at the same time: these two searches will be referred to in this paper as the *keyword serial* and the *keyword parallel* search respectively. As Gudes and Ganesh[7] note, the keyword parallel algorithm is rather more complex in character but is much more efficient in operation when sorted multilists are used since the records may be processed in sequence, substantially reducing the amount of disc seek time for files that extend over more than one cylinder. The new best match search procedure described below, which is based on Hsiao and Harary's keyword parallel algorithm, has been developed with this factor very much in mind. Unsorted multilists result in random accesses to the discs with consequent delays while the disc heads move from one track to another: efficient algorithms for partial match retrieval from unsorted multilists are described by Claybrook and Yang[9]. Even if the multilists are ordered, keyword serial list searching algorithms will involve the data file being scanned several times, once for each of the query term lists, and will again occasion a considerable amount of head movement.

It should be noted that the full benefits of keyword parallel searching, in terms of minimizing disc head movement, will be obtained only if the search algorithm is implemented using a disc unit that is dedicated to servicing a particular search until it has been completed[9]: an example of such an implementation would be a single user microcomputer system. Conversely, if the disc unit is shared between several processes that are executing simultaneously, as with the time-sharing system used for the experiments reported here, it is likely that a considerable degree of head contention will ensue. In this case, keyword parallel searching is unlikely to offer any advantages unless some form of operating system control is adopted to minimize head movement[10]. Similar considerations apply to sorting algorithms that are designed to minimize disc head movement, as with that described by Cooper *et al.*[11]

In the discussion that follows, it is assumed that each query, Q, is represented by a binary term vector

$$(q_1, q_2 \ldots q_i \ldots q_m)$$

where q_i is the ith indexing term to have been assigned to the query. When the entries in the directory corresponding to these terms are inspected, a comparable vector of identifiers

$$(\mathrm{FIRST}[q_1], \mathrm{FIRST}[q_2] \ldots \mathrm{FIRST}[q_i] \ldots \mathrm{FIRST}[q_m])$$

is obtained where $\mathrm{FIRST}[q_i]$ represents the first document containing q_i. A document in the datafile has associated with it a term vector

$$(d_1, d_2 \ldots d_j \ldots d_n)$$

and a similar vector of identifiers

$$(\mathrm{NEXT}[d_1], \mathrm{NEXT}[d_2] \ldots \mathrm{NEXT}[d_j] \ldots \mathrm{NEXT}[d_n])$$

where $\mathrm{NEXT}[d_j]$ is the identifier of the next document containing the jth document term: this identifier may have the value NULL if there are no further documents to which that term has been assigned.

Keyword serial best match search algorithm

A keyword serial search algorithm for multilist files may be obtained by the use of the upperbound procedure for inverted files described by Smeaton and van Rijsbergen[1]. The query terms are sorted into increasing frequency order so that the shortest lists of document identifiers are processed first: such frequency information is normally included in the directory of a multilist file. In addition, a table, MIN, is created that contains the numbers of terms in the shortest document, i.e., the document containing the smallest number of terms, in each of the lists of documents in the directory: thus $\mathrm{MIN}[q_i]$ gives the length of the shortest document containing the ith query term. The MIN value of such a term could sensibly be included in the corresponding directory entry, with the value being updated to reflect changes in the constitution of the list.

Assume that i of the m query terms have been processed so that an uninspected document can have at most $m - i$ terms in common with the query. Knowing the number of terms in the query, m, and in the shortest document in the $i + 1$th list, $\mathrm{MIN}[q_{i+1}]$, the maximum possible similarity for documents in that list can be calculated, the exact form of this upperbound calculation depending upon the particular similarity coefficient that is being employed. If the upperbound similarity is less than that for the current nearest neighbour, none of the documents in the list needs to be inspected; since the query terms are processed in order of increasing collection frequency it will be the longer lists that will be eliminated by this procedure. Fuller details of the algorithm are given by Smeaton and van Rijsbergen[1] in their original paper, while Perry and Willett[4] describe a modification that results in a slightly more precise upperbound calculation.

The only difference between the inverted file and multilist implementations of the algorithm is in the manner in which the next document for processing is

determined, this difference arising from the two types of file structure: the upperbound calculations and reductions in file search are identical in the two cases. There is, however, one area where the multilist file structure necessitates a modification of the original algorithm. This change arises from the fact that if a document has more than one term in common with the query, its identifier will appear in more than one of the lists corresponding to the terms in the query. Some of these lists may be obviated *in toto* by the upperbound procedure but, for those lists which are processed, the documents will need to be retrieved from backing storage whenever they have been assigned the corresponding term. Since a backing storage access may be as time-consuming as many thousands of main store similarity calculations, it is vital to eliminate such redundant accesses if at all possible, and Smeaton and van Rijsbergen describe a means by which this may be achieved in the inverted file context. A note is kept of all those documents processed during a search so that if a previously encountered document identifier is found in the current inverted file list, no attempt is made to retrieve the corresponding document from backing storage; instead, consideration is transferred to the next identifier in the inverted file list. Such an approach is not appropriate to multilist processing since the next document can only be identified from the pointers associated with the current one, i.e., the previously encountered document. It would thus seem that this document must be retrieved again if processing is to continue, causing a redundant, and expensive, disc access. Such accesses may, however, be obviated if the list of previously encountered documents is augmented by the inclusion of some additional information, as suggested by Yang[12] in the context of partial match retrieval from multilists. Specifically, when a document, D, is encountered that has more than one term in common with the query, a note is made in an internal table of the other query terms occurring in the document, together with the corresponding identifiers NEXT[q_j], NEXT[q_k] etc. During the processing of a subsequent multilist, q_j say, the occurrence of the identifier for D as the next document triggers an access not to backing storage, but to the internal table to determine the document after D. The reduction in such redundant disc accesses will rise in line with the mean number of terms in common between a query and those documents with which it has at least one term in common.

Keyword parallel best match search algorithm

The algorithm to be described here provides a means of traversing the m lists associated with some query in parallel without the need to inspect all of the documents in these lists, while still ensuring the identification of the nearest neighbour. The algorithm involves the maintenance of an array, **S**, which contains the identifiers of the current documents in each of the m keyword lists. **S** is initialized by the set of identifiers FIRST[q_1], FIRST[q_2] . . FIRST[q_m] obtained by an inspection of the keyword directory. Let **T** be another array containing the identifiers in **S** after they have been sorted into ascending order and duplicates eliminated, while the array **F** contains the corresponding frequencies of occurrence in **S**: thus **T**[1] will represent the lowest document identifier in **S**, one which occurs **F**[1] times there. BESTCOEFF contains the highest similarity coefficient at any point in the search; BESTCOEFF is initially set to zero in the case of a similarity coefficient, or to a large value in the case of a dissimilarity or distance function.

The heart of the search algorithm is an upperbound procedure for the selection of the document in **S** that must next be retrieved from backing storage for matching against the query. As each such document is identified, **S** is updated and the procedure iterated until all of the m keyword lists have been traversed, or until it is certain that the best match has been identified.

Having initialized **S** and the elements of **T** and **F**, the question arises as to whether there is a document with an identifier, a, such that $\mathbf{T}[1] <= a < \mathbf{T}[2]$, and which could have a greater similarity to the query than BESTCOEFF. The maximum possible number of terms, MAX, that the document can have in common with the query is **F**[1] since, if it had more, it would have appeared in **S**. Knowing MAX and the length of the shortest document vector amongst those documents containing the term, this information coming from MIN, as in the keyword serial best match search described above, an upperbound to the similarity may be calculated for documents in the range **T**[1] to **T**[2] − 1. If the upperbound is greater than BESTCOEFF then documents in this range must be inspected since it is possible that one of them is the best match. Such documents, if in fact they exist, may be processed by taking as the first document which needs to be retrieved that whose identifier is contained in **T**[1]. If, however, no such potential best match exists, consideration passes to those documents with addresses in the range $\mathbf{T}[2] <= a < \mathbf{T}[3]$. As before, an upperbound similarity may be calculated for the documents: in this case, MAX is given by **F**[1] + **F**[2]; in general, for documents with identifiers in the range $\mathbf{T}[x] <= a < \mathbf{T}[x+1]$, MAX is given by $\sum_{y=1}^{x} \mathbf{F}[y]$ and the document length chosen for the upperbound calculation is the smallest of the entries in MIN corresponding to these terms.

This procedure for the evaluation of documents in the ranges **T**[x] to **T**[$x+1$] − 1 for $x = 1, 2$ . . continues until some document is identified whose upperbound similarity is greater than BESTCOEFF: the identification of such a potential best match results in the selection of **T**[x] as the next document that must be processed. Once this document has been identified, it is retrieved from backing storage and the similarity calculated and compared with BESTCOEFF, which is modified if appropriate. Attention is then given to the pointers associated with the query terms that occur in this document. For each such term, q_i, present, the corresponding identifier NEXT[q_i], or NULL, is used to update the entry corresponding to q_i in the set **S**. Once this updating has been completed, the process of selecting the next document for inspection may be re-commenced by identifying the elements of **T** and **F**. An example of the workings of this algorithm is given as an appendix to this paper.

It should be noted that it is possible for no potential nearest neighbour to be found. In this case, the upperbound calculation may be performed for documents lying between the last element of **T**, **T**[m'], and N, the end of the file, with MAX set to m; if a possible best match is then identified, **T**[m'] may be selected as the next

document for processing, otherwise, the search may be terminated.

EXPERIMENTAL DETAILS AND RESULTS

The experiments involved the three well-known document test collections detailed in Table 1. Although the collections are quite small, they are sufficiently disparate in character to provide a fair assessment of the efficiency of the search algorithm.

Table 1. Details of the three test collections used

	Keen	Cranfield	Evans
Number of documents	800	1400	2542
Number of queries	63	225	39
Number of terms	1432	2557	3730
Mean number of terms per document	9.8	28.7	6.6
Mean number of terms per query	10.3	8.0	27.5

Five different measures were used to determine the degree of similarity between the query and a document in the file, these being the Simple, Dice, Cosine and Overlap similarity coefficients, and the Hamming distance. For a query containing m terms, and a document containing n terms, c of which are common to both, these coefficients are defined as

Simple: c
Dice: $2c/(m + n)$
Cosine: $c/\sqrt{m*n}$
Overlap: $c/\min(m, n)$
Hamming: $2c - m - n$

In the case of the Hamming distance, it is assumed that a document *must* have at least one term in common with the query for it to be considered as a nearest neighbour[2].

The experimental measure used to evaluate the efficiency of the search algorithms is the mean fraction of each file that needed to be retrieved and searched, the average being taken over the entire set of queries in each case. Table 2 contains the search results obtained using the keyword parallel algorithm, while Table 3 lists the corresponding results for the keyword serial algorithm.

The search measure ignores any additional storage and processing overheads resulting from the use of the algorithms. In the case of the keyword serial search, the use of the internal table to obviate the retrieval of previously inspected documents may be efficiently implemented as described by Yang[12], who has shown that

Table 2. Search efficiency using the keyword parallel best match algorithm

	Keen	Cranfield	Evans
Simple	0.12	0.11	0.19
Dice	0.25	0.31	0.21
Cosine	0.27	0.30	0.22
Overlap	0.28	0.18	0.17
Hamming	0.29	0.33	0.22

Table 3. Search efficiency using the keyword serial best match algorithm

	Keen	Cranfield	Evans
Simple	0.06	0.05	0.13
Dice	0.11	0.21	0.15
Cosine	0.13	0.19	0.16
Overlap	0.16	0.08	0.16
Hamming	0.17	0.28	0.17

the overheads are not onerous, and are likely to be of little consequence in comparison with the reduction in disc accesses that results. In the case of the keyword parallel search, on the other hand, space must be allocated for the arrays **S**, **T** and **F**, and these must be updated as a search proceeds; however, these arrays are of size m elements or less for a query containing m terms and the overheads are again small since, as Table 1 shows, queries typically contain only relatively small numbers of terms.

In a conventional nearest neighbour search, each of the documents must be matched against the query to find the most similar one(s) in the file: in this case, the mean fraction of the file searched is clearly 1.0. A superior search may be obtained by traversing the multilists corresponding to the query terms and matching only the documents in these lists against the query, thus removing from consideration the many documents that share no common terms with the query. This traversal may be accomplished by using either of Hsiao and Harary's search algorithms, and both of them will give the same reduction in search, if some procedure is adopted for the elimination of duplicate documents as described earlier. The mean fractions of each file that would need to be inspected in such a search are 0.37, 0.52 and 0.24 for the Keen, Cranfield and Evans data sets, respectively, and these figures may be used as a baseline against which the keyword serial and keyword parallel best match algorithms may be compared. It should be noted that the figures are identical for all of the similarity measures listed above since the reduction in search is determined only by the documents present in the query terms multilists, and is independent of the particular similarity measure that is chosen.

DISCUSSION

It will be seen from the results presented in Table 2 that the proposed algorithm can achieve substantial reductions in the numbers of documents that need to be retrieved when compared with the numbers retrieved in a conventional multilist search. The reductions vary from collection to collection, this reflecting the variant frequency characteristics of the three data sets studied[4]. In particular, the Evans collection shows only a very small reduction in the expected search length in comparison with the conventional keyword parallel multilist search. This arises because of the large numbers of terms that are used to characterize each of the queries in this test set; with exhaustive queries, each of the ranges **T**[1] to **T**[2] − 1, **T**[2] to **T**[3] − 1, etc., will be short and thus relatively few documents will be obviated by the upperbound calculation. When the ranges are greater, as is the case with the Keen and Cranfield collections where there are fewer terms associated with each query, the

number of documents eliminated in each iteration of the search is likely to be proportionally greater.

The search lengths obtained with the various normalized coefficients, i.e., the Dice, Cosine, Overlap and Hamming measures, do not vary very much within a data set, except in the case of the Overlap coefficient with the Cranfield and Evans test sets. This behaviour arises from the asymmetric nature of the coefficient, which involves the length of either the document or the query, whichever is the shorter, whereas the other measures involve both of these lengths. For the Cranfield and Evans data, there is a great disparity in the document and query lengths and this is reflected both in the observed and upperbound similarities and in the numbers of documents eliminated from the search. A detailed account of the subtle interplay between the type of similarity measure used, the characteristics of the test collection, and the relative magnitudes of the observed and upperbound similarities is given by Perry[13] in the context of inverted file searching.

Table 3 contains the results for the keyword serial best match search. This search is markedly more efficient in terms of the numbers of similarity calculations since entire lists of document identifiers may be eliminated from consideration by the upperbound procedure. In the keyword parallel best match search, conversely, only sublists, some of which contain only a very few documents, are eliminated at each stage in the search, and the savings are proportionally less. As noted earlier, however, the latter algorithm does ensure that the documents may be retrieved from backing storage in a single linear scan.

Murtagh[2] described a modification of the Smeaton–van Rijsbergen algorithm for nearest neighbour searching in inverted files. The modification involved the lengths of individual documents in the calculation of the upperbounds, rather than the use of the MIN values as in the experiments reported here. Murtagh found that this modification led to large increases in search efficiency over the basic algorithm, and the question arises as to whether such a procedure may be adopted for multilist searching. In the case of keyword parallel searching, let there be an internal table, DOCLENGTH, which contains the numbers of terms assigned to each of the documents in the collection. Then, when documents with identifiers in the range $\mathbf{T}[x] <= a < \mathbf{T}[x+1]$ are being considered, an upperbound may be calculated for the similarity of *each* document in this range by using the calculated value for MAX in conjunction with the corresponding element from DOCLENGTH. If any of the upperbounds are greater than BESTCOEFF then a nearest neighbour may be present and $\mathbf{T}[x]$ must be retrieved. Such an approach has the advantage that fewer documents will need to be retrieved from backing storage since more accurate upperbounds are calculated. In fact, experiments using such a procedure yielded reductions in search length of up to 25 per cent when compared with the figures presented in Table 2. There are, however, two major problems associated with such an approach.

Firstly, although the number of *actual* calculations is considerably less than with the keyword parallel algorithm presented earlier, the number of *upperbound* calculations is very much greater: for documents in the range $\mathbf{T}[x] <= a < \mathbf{T}[x+1]$, the MIN method requires only a single upperbound calculation, whereas the DOCLENGTH method would involve one for each of these documents. Internal calculations are, of course, very much faster than accesses to backing storage but, for large files, the use of the DOCLENGTH upperbound would necessitate very many upperbound calculations indeed. Secondly, and more importantly, the DOCLENGTH method sometimes results in a non-monotonically increasing sequence of document identifiers, i.e., it may happen that a document needs to be retrieved from backing storage for matching against the query even though its identifier is less than that of a document that had been retrieved previously in the search. Such an occurrence, which was noted with the test sets used here, interrupts the linear scan of the disc that is the *raison d'etre* for keyword parallel retrieval. To see why this might be so, consider some document with identifier $\mathbf{T}[1]$ and suppose that the upperbound calculation resulted in $\mathbf{T}[2]$ being selected as the next document that needs to be retrieved; **S**, **T** and **F** are recalculated, this resulting in some new value for $\mathbf{T}[2]$, $\mathbf{T}[2']$ say. If it happens that there is some document lying in the range $\mathbf{T}[2]$ to $\mathbf{T}[2'] - 1$ and having a small DOCLENGTH value, it may well be that the new set of upperbound calculations results in the identification of a potential nearest neighbour in the range $\mathbf{T}[1] <= a < \mathbf{T}[2']$; hence $\mathbf{T}[1]$ will be the next document that must be retrieved, despite the fact that it corresponds to a document occurring earlier in the file than $\mathbf{T}[2]$ which had been retrieved in the previous iteration of the algorithm.

The Murtagh algorithm is also, unfortunately, not applicable to the keyword serial search, owing to the nature of the multilist file structure. In the case of an inverted file search, the identifiers for the documents in a list are present within the main store and, given the table DOCLENGTH, the upperbounds may be easily calculated; the full document representatives are then retrieved from backing storage only if the upperbound similarity is greater than BESTCOEFF. With keyword serial multilist searching, the next document in a list must be brought into the main store, irrespective of its upperbound or actual similarity with the query, to enable the identification of subsequent members of that list; an upperbound calculation may, of course, then be performed but the great bulk of the computational expense will already have been incurred in the backing storage access. Similar comments apply to the efficient inverted file nearest neighbour procedure described by Noreault *et al.*[14] and by Perry and Willett[4].

In conclusion, it is clear that the efficiency of nearest neighbour searching in multilists is very much less than the best that may be achieved in the case of inverted files. The keyword serial search can perform only as well as the Smeaton–van Rijsbergen algorithm, and this is achieved when a fair amount of additional processing is carried out to obviate the retrieval of duplicate documents. The keyword parallel algorithm is still less efficient in terms of the numbers of documents retrieved; however, this algorithm does not need to invoke a procedure for handling duplicate documents, and it may also involve rather less movement of the disc heads in some circumstances.

ACKNOWLEDGEMENTS

My thanks are due to Dr K. Sparck Jones and Mr L. Evans for provision of the datasets used in this study, and to Prof M. F. Lynch, Mr I. G. Hendry and the referees for comments on earlier drafts of this paper.

REFERENCES

1 **Smeaton, A F and van Rijsbergen, C J** 'The nearest neighbour problem in information retrieval. An algorithm using upperbounds" *ACM SIGIR Forum* Vol 16 (1981) pp 83–87

2 **Murtagh, F** 'A very fast, exact nearest neighbour algorithm for use in information retrieval' *Info. Tech.* Vol 1 (1982) pp 275–283

3 **Eastman, C M and Weiss, S F** 'Tree structures for high dimensionality nearest neighbour searching' *Info. Syst.* Vol 7 (1982) pp 115–122

4 **Perry, S A and Willett, P** 'A review of the use of inverted files for best match searching in information retrieval systems' *J. Info. Sci.* Vol 6 (1983) pp 59–66

5 **Lefkowitz, D** *File Structures for On-Line Systems* Spartan Books, New York (1969)

6 **McDonell, K J** 'The design of associative key lists (secondary indices)' *Aust. Computer J.* Vol 8 (1976) pp 13–18

7 **Gudes, E and Ganesh, S** 'A survey of file organizations and performance' *Adv. Info. Syst. Sci.* Vol 8 (1981) pp 1–73

8 **Hsiao, D K and Harary, F** 'A formal system for information retrieval from files' *Commun. ACM (USA)* Vol 13 (1970) pp 67–73

9 **Claybrook, B G and Yang, C S** 'Efficient algorithms for answering queries with unsorted multilists' *Info. Syst.* Vol 3 (1978) pp 93–97

10 **Teory, T J and Pinkerton, T B** 'A comparative analysis of disc scheduling policies' *Commun. ACM (USA)* Vol 15 (1972) pp 177–186

11 **Cooper, D, Dicker, M E and Lynch, M F** 'Sorting of textual data bases: a variety generation approach to distribution sorting' *Info. Proc. Manage.* Vol 16 (1980) pp 49–56

12 **Yang, C S** 'Avoiding redundant record accesses in unsorted multilist file organizations' *Info. Syst.* Vol 2 (1977) pp 155–158

13 **Perry, S A** *Inverted file, nearest neighbour search algorithms for document retrieval* MSc dissertation, University of Sheffield (1982)

14 **Noreault, T, Koll, M and McGill, M J** 'Automatic ranked output from Boolean searches in SIRE' *J. Am. Soc. Inf. Sci. (USA)* Vol 28 (1977) pp 333–339

APPENDIX

The operation of the algorithm will be illustrated by reference to a simple query containing five terms $q_1 - q_5$. Assume that MIN contains the values

(3, 4, 3, 6, 2)

as might be the case with best match retrieval from a file of document titles, and let BESTCOEFF be initialized to zero prior to a best match search using the Dice coefficient as the similarity measure. Let **S** contain

(26, 13, 57, 26, 82)

so that **T** and **F** contain

(13, 26, 57, 82) and (1, 2, 1, 1)

respectively. Then the upperbound for documents with identifiers in the range 13 to 25 is given by 2*1/(4 + 5), which is about 0.222: this is greater than the current value in BESTCOEFF, zero, so document number 13 is retrieved, the actual coefficient calculated, and BESTCOEFF updated. After inspection of the NEXT pointers for this document, **S** is updated to

(26, 26, 57, 26, 82)

which results in

(26, 57, 82) and (3, 1, 1)

for **T** and **F**. The upperbound for documents in the range 26 to 56 is given by evaluating 2*3/(min(3, 4, 6) + 5); as this upperbound value is greater than that currently stored in BESTCOEFF, document number 26 is retrieved and the actual coefficient evaluated: let the calculated value be 0.47. Then BESTCOEFF is modified to reflect the new nearest neighbour, and **S** updated, this resulting in **S, T** and **F** now containing

(30, 82, 57, 118, 82), (30, 57, 82, 118) and (1, 1, 2, 1)

respectively. The upperbound similarity for documents with identifiers in the range 30 to 56 is 2*1/(3 + 5) which is less than the value currently stored in BESTCOEFF so all of these documents may be ignored. The comparable upperbound for the second group of identifiers is 2*2/(3 + 5) which is greater than BESTCOEFF: accordingly, document number 57 would be chosen as the document that must be processed prior to the next iteration of the search.