The Efficiency of Inverted Index and Cluster Searches

Ellen M. Voorhees

TR 86-751
May 1986

Department of Computer Science
Cornell University
Ithaca, NY 14853

The Efficiency of Inverted Index and Cluster Searches

Ellen M. Voorhees

Cornell University

Abstract: The processing time and disk space requirements of an inverted index and top-down cluster search are compared. The cluster search is shown to use both more time and more disk space, mostly due to the large number of cluster centroids needed by the search. When shorter centroids are used, the efficiency of the cluster search improves, but the inverted index search remains more efficient.

1. Introduction

Research by Griffiths *et al.* suggests that the single link clustering method is not as effective as other agglomerative, hierarchic clustering methods for document retrieval [GRIF84, GRIF85]. Recent research by the author corroborates these findings and also shows that the partial ranking produced by a top-down search of the complete link hierarchy can be more effective than ranking the collection by decreasing similarity to the query. Furthermore, the evidence suggests that searches of the complete link hierarchy will remain effective independent of collection size [VOOR86].

However, in order to be useful a search must be efficient as well as effective. This paper compares the efficiency of the cluster search to the efficiency of an inverted index search that ranks the entire collection. The efficiency of each search is measured in two ways: the number of bytes required to store the auxiliary files required by the search, and the mean time required to retrieve a set of documents for a query. The processing time is further divided into CPU time and I/O time. The measurements are obtained by observing searches on four test collections.

The results of the experiments indicate that the inverted file search is more efficient in terms of both space and processing time since the cluster search requires many centroid vectors. The maximum length of a centroid vector can be significantly decreased before the effectiveness of the cluster search suffers, but the efficiency increase gained in this way is not sufficient to make the cluster search as efficient as the inverted index search.

2. The Searches

The purpose of the inverted index and cluster searches is to return a set of documents to the user in response to a query. The documents returned are the documents encountered in the search that have the highest similarities to the query. In the current experiments, the documents and queries are represented by weighted vectors and the inner product similarity is used. The weight of a term in a vector is proportional to the number of times the term appears in the document or query and inversely proportional to the number of documents in which it appears. The weights in each vector are normalized so that the inner product similarity of two vectors is equal to their cosine similarity.

2.1. The Inverted Index Search

The inverted index search encounters all documents with non-zero similarities to the query, and thus is guaranteed to return the documents with the highest similarities. The search requires an inverted index – a representation of the collection in which access to the collection is made through terms instead of through documents. Associated with each term in the collection is a list of <document id,weight> pairs for each document in which that term appears. The inverted index search manipulates the inverted lists of query terms as described below.

The search used here is the 'basic algorithm' described by Buckley and Lewit [BUCK85]. The search requires enough internal memory space so that a partial similarity can be accumulated between the query and each document (*e.g.* 4N bytes where N is the size of the collection). Another array contains the best x partial similarities seen so far (where x is the number of documents to be returned to the user). The search begins by setting each of the partial similarities to zero. The inverted index list for each query term is read in turn. For each document, d, on a list, the product of the weights of the current term in the query and in d is added to d's partial sum. If d's current similarity is large enough, it is placed in the top similarities array. After all query term lists have been processed, the top similarity array contains the documents to be returned to the user. Figure 1 shows a sample search for a query consisting of terms A, B, and C when two documents are to be returned.

2.2. The Cluster Search

Cluster searches were introduced as both a means of improving the efficiency [SMAR71] and the effectiveness [JARD71] of a retrieval system. Cluster searches do not encounter all documents, and therefore, they are not guaranteed to retrieve the documents with the greatest similarities to the query. The particular search used here requires several auxiliary files: the representation of the cluster hierarchy, the centroid file, and the document file.

The hierarchy used in these experiments is the complete link hierarchy. This hierarchy is formed by repeatedly merging the clusters with the maximum similarity, where the similarity between clusters is defined to be the minimum similarity between any pair of documents in the clusters. Due to the strict clustering criterion, complete link hierarchies tend to be quite shallow and bushy. The content of each cluster is summarized by a centroid vector; the similarity between a query and a cluster is computed using its centroid. The centroids in these experiments were created as follows:

- The sum of the within document frequency of each term in the cluster is computed, and the terms are sorted by decreasing frequency.
- The top 250 terms are selected to be in the centroid. The weight of each of the terms in the centroid is the rank (from the bottom) of the term in the sorted list – equal frequencies are assigned the same rank.

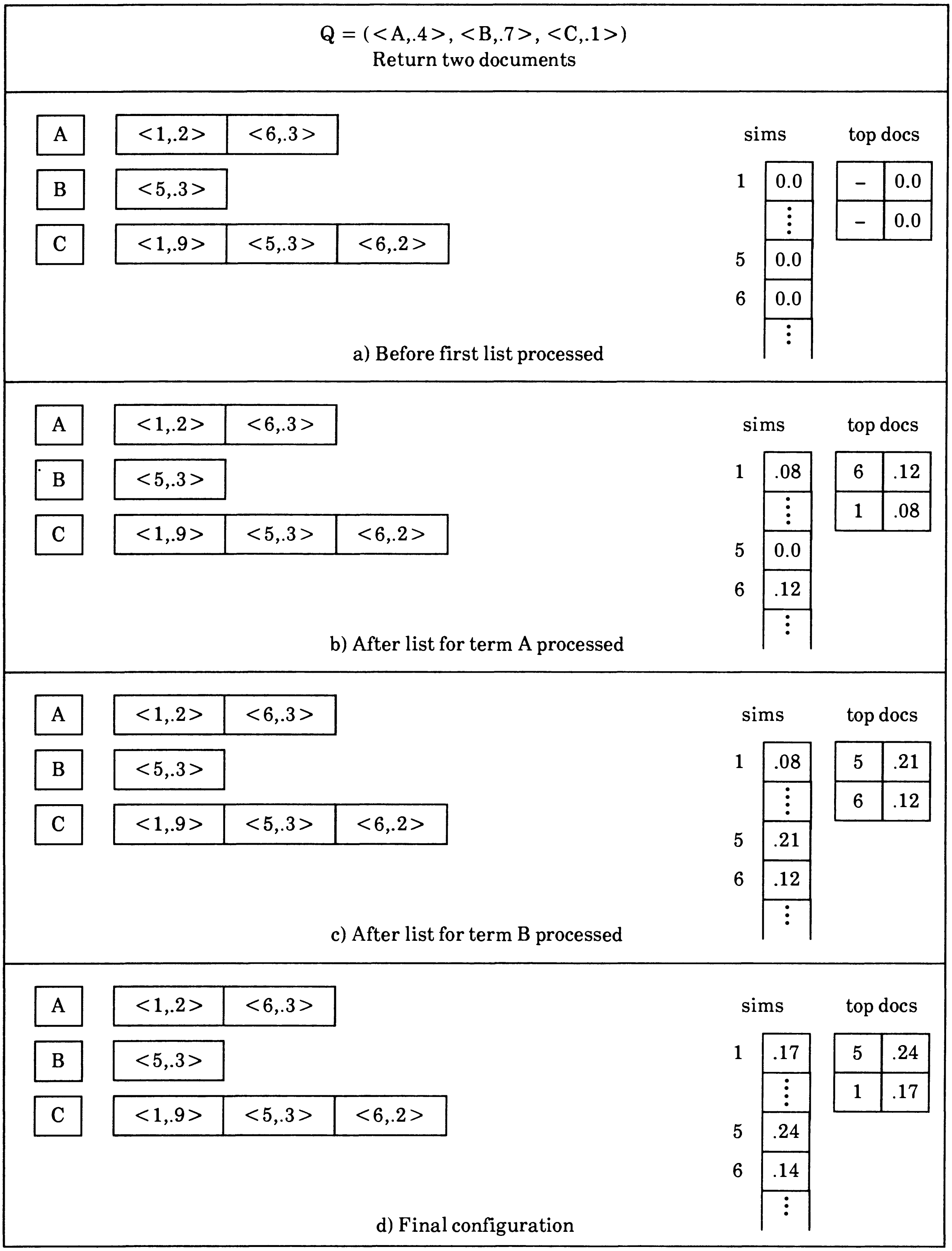


Figure 1: Sample Inverted Index Search

- The rank weights are multiplied by an inverse document frequency factor and normalized so the sum of the squares of the weights equals one (cosine normalization).

The number of clusters that are encountered in the search depends on the parameter *NumWanted*. This parameter specifies the number of documents that should be retrieved by the search. In general, retrieving more documents will be more effective but also more expensive than retrieving fewer documents.

The search begins by placing the root of the hierarchy into an empty heap. The top of the heap (corresponding to the cluster with the current largest similarity with the query) is popped until sufficient documents have been retrieved or the heap is empty. If the hierarchy node popped from the heap represents a document, that document is retrieved. Otherwise, all the children of the node that have non-zero similarities with the query are added to the heap. When sufficient documents have been retrieved, the retrieved set is ranked by decreasing similarity.

Since the path down the tree is controlled by the nodes that are popped from the heap, the search is neither a true depth-first nor a true breadth-first search of the hierarchy. In practice, it is closer to a depth-first search with some backtracking than to a breadth-first search.

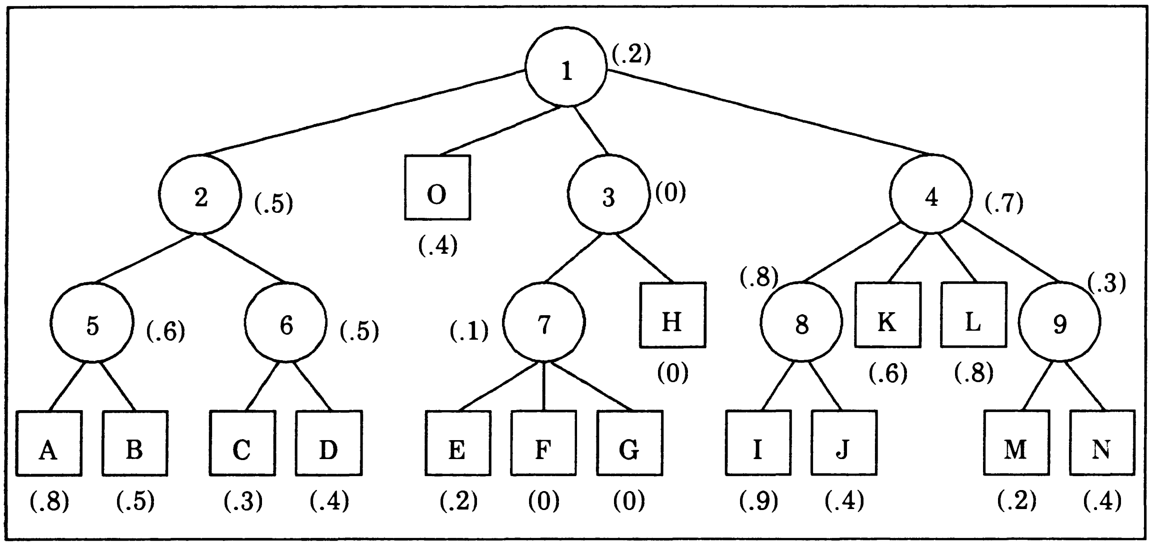
A sample top-down search for a hypothetical hierarchy and query is given in Figure 2. In the Figure, the squares represent documents and the circles represent non-singleton clusters. The numbers within the circles are node numbers; the documents are identified by letters. The number in parentheses next to a node is the similarity of that node with the query.

3. Storage Requirements

This section investigates the amount of external memory space required to store a document collection and the data structures needed to search it. (A disk is the secondary storage device assumed in the rest of the paper.) The size is measured in the number of bytes required to store the data assuming the data structures are implemented as described below. The storage required for the natural language text of the documents is not included in the comparisons since the same amount of storage is used for the text regardless of the searching strategy.

A vector, either a document vector or a centroid vector, consists of a fixed length header and an arbitrary number of tuples. The header contains an id number and the number of tuples in the vector (8 bytes). Each tuple contains a concept number and the weight of that concept in the vector (8 bytes).

An inverted index entry has a similar structure. The header contains the concept number, the number of documents in which the term appears, and a pointer to tuples that make up the rest of the entry (12 bytes). Each tuple contains the vector id and the weight of the concept in that vector (8 bytes). There is an inverted index entry for each distinct concept in the collection.



a) Hierarchy to be searched

add[1,.2]	heap: 1
pop[1,.2]	
add[2, .5], [4, .7], [O, .4]	heap: 2, 4, O
pop[4, .7]	
add[8, .8], [9, .3], [L, .8], [K, .6]	heap: 2, O, 8, 9, L, K
pop[L, .8]	1 document retrieved
pop[8, .8]	
add[I, .9], [J, .4]	heap: 2,O,9,K, I, J
pop[I, .9]	2 documents retrieved
pop[K, .6]	3 documents retrieved
	STOP
	Retrieved Documents:
	I sim = .9
	L sim = .8
	K sim = .6

b) trace of search with *NumWanted* = 3

Figure 2: Sample Top-down Cluster Search

Table 1: Collection Characteristics

	CACM	CISI	INSPEC	MED
number docs	3204	1460	12684	1033
number terms	8503	4941	14573	6927
mean terms per doc	22.5	43.9	32.5	51.6
number queries	50	35	77	30
mean relevant docs per query	15.8	49.8	33.0	23.2
total relevant docs	792	1742	2543	696
exhaustive rel. assessments	no	yes	no	yes

The hierarchy is described by a set of fixed length nodes, one node for each cluster (including singleton clusters) in the hierarchy. A node contains six fields: the node id, the similarity value at which the cluster it represents was formed, a pointer to the vector associated with the node, the number of children, the node id of its first child, and the node id of its rightmost sibling (24 bytes).

3.1. Disk Usage Experiments

The inverted index search needs to access only the inverted file in order to complete a search. However, to perform some other retrieval functions such as query expansion for relevance feedback, a direct file of the document vectors is also needed. The space measured for the inverted index search, therefore, is the amount of space used by both the direct and inverted files. The space measured for the cluster search is the amount of space required by the hierarchy file, the centroid file, and the direct document file. The maximum length of a centroid is restricted to 250 terms in these experiments.

The auxiliary files needed for the inverted index and cluster searches were created for four test collections: CACM*, a computer science collection; CISI*, a library science collection; INSPEC, an electrical engineering collection used by permission of The Institution of Electrical Engineers, Hertfordshire, England; and MED, a bio-medicine collection. Details of these collections can be found in Table 1. The size of each auxiliary file and the total size for each search is given in Table 2. The percentage increase of the cluster search over the inverted index search is also included in the Table.

The numbers in Table 2 show that the cluster search requires a significant amount more disk space than the inverted index search (an increase of 70-100%). The cluster search requires larger amounts of

* The CACM and CISI collections contain information other than standard content indicators [FOX83]. However, only the standard content indicators are used in these experiments. As a result, two queries have been removed from the CACM query collection since they contained only non-content indicators.

Table 2: External Storage Space in Bytes for Inverted Index and Cluster Searches (including % increase of the cluster search over the inverted index search)

	doc vectors	inverted index	total for inverted	hierarchy file	centroid file	total for clusters	% increase
CACM	603,488	679,892	1,283,380	142,392	1,481,320	2,227,200	74
CISI	539,584	572,084	1,111,668	67,320	1,375,536	1,921,850	73
INSPEC	3,399,512	3,472,916	6,872,428	589,488	9,685,704	13,674,704	99
MED	434,656	509,516	944,172	47,736	1,132,544	1,614,936	71

disk space due to the number of centroids that need to be stored – the centroid file is larger than the inverted index for each of the collections.

Secondary storage is relatively inexpensive; a more critical factor of the efficiency of a search is the amoünt of time it takes to retrieve a set of documents. This topic is discussed in the next section.

4. Processing Time Requirements

As was mentioned in the introduction, the total time to process a query is assumed to be the sum of the CPU time and the I/O time. The CPU time is the time the main processor devotes to retrieving documents for the query. The I/O time is the amount of time the search spends waiting for pages to be retrieved from the disk.

4.1. CPU Time

The CPU time is measured by the number of idealized machine instructions performed in the search. Each (floating point) addition, (floating point) multiplication, comparison, pointer increment, and array indexing is counted as one operation. The instructions are counted only in the main part of each search (*i.e.* no initialization costs are included). For the inverted index search, Figure 3(a), the main loop is the loop which processes each inverted list. Five machine instructions are required for each document id in the lists. For the top-down search, Figure 3(b), the main part is computing the similarity between a given query and document or centroid vector. The cluster search performs three instructions for each term that appears in exactly one of the query and centroid (or document) vectors and six instructions when a match occurs.

4.2. I/O Time

The amount of information that needs to reside in core at any given time is not very large for any of the searches. Assuming the inverted lists contain the weights of the terms in the documents, the inverted index search requires only the inverted lists of each query term. The top-down search needs to keep the hierarchy nodes that have been visited and the current centroid or document vector in core. Each vector is needed only once (if it is needed at all) so the same memory location can be used for each vector. The

```
for (each query term list) {
    if (at end of list) {                                   (1 instruction)
        go to next list;
    }
    sims[current did] + = query term wt * doc term wt;      (3 instructions)
    increase list ptr to point to next did in list;         (1 instruction)
}
```

a) inverted index search

```
sim = 0;
while (true) {
    if (query concept num > doc concept number) {           (1 instruction)
        increase doc ptr;                                   (1 instruction)
        if (at end of doc) {                                (1 instruction)
            exit loop;
        }
    }
    else if (query concept num < doc concept number) {      (1 instruction)
        increase query ptr;                                 (1 instruction)
        if (at end of query) {                              (1 instruction)
            exit loop;
        }
    }
    else {
        sim + =query term wt * doc term wt;                 (2 instructions)
        increase query ptr;                                 (1 instruction)
        if (at end of query) {                              (1 instruction)
            exit loop;
        }
        increase doc ptr;                                   (1 instruction)
        if (at end of doc) {                                (1 instruction)
            exit loop;
        }
    }
}
```

b) cluster search

Figure 3: Idealized Instruction Counts for Both Searches

hierarchy nodes may be accessed more than once, so they must be stored in separate locations. Since each node consists of only 24 bytes and the top-down search is closer to a depth-first search than a breadth-first search, the amount of space required by the hierarchy nodes is not excessive. For example, the mean number of nodes accessed in the top-down search of the complete link hierarchy for the INSPEC collection is 873 (requiring only 20,952 bytes of core). These memory usage considerations do not include the memory needed by the program variables such as the array of similarities for the inverted index search and the heap used to store the similarities in the cluster search.

Any modern computer will have enough memory to meet these demands, and thus no paging strategy is considered in the experiments. That is, it is assumed that if a page that has already been brought into main memory for this query is accessed again, the page still resides in core. The I/O time of a search, therefore, is proportional to the number of data pages that are accessed at least once, and this figure is used as the measure of the I/O cost of a search. This measure makes the unrealistic assumption that each page access takes the same amount of time. In fact, the time necessary to access a page is the sum of the seek time, the rotational delay, and the data transfer times, so some page accesses are less expensive than others. However, the experiments do not use this level of detail.

Another implicit assumption in this measure is that the cost of presenting the same number of documents as output is approximately equal for the two searches. This assumption is also untrue: the natural language text for the top-down search can be stored in cluster order while the text for the inverted index search is stored randomly. Clustering the text should cause fewer disk accesses to be required since documents on the same page are likely to be retrieved together. Furthermore, the cost of accessing sequential pages is less than accessing the same number of random pages. However, due to the small number of documents retrieved, this difference is negligible compared to the difference in the number of disk accesses required in the searches themselves.

In order to count the number of pages accessed during a search, some allocation of the data to pages must be assumed. The model of page allocation used in these experiments is a compromise between making the searches as efficient as possible and accurately modeling an operational environment. Collection maintenance – retiring old documents and adding new documents – is an important consideration in an operational environment. Thus the model cannot assume the pages are allocated in such a way that the efficiency would seriously deteriorate or the entire collection would need to be reconfigured if new documents are added to the collection.

The page length is assumed to be 4096 bytes. Inverted index headers are packed into as few pages as possible without splitting a header across page boundaries. Each list starts on a new page and takes as few pages as possible without splitting a tuple across page boundaries. Starting each list on a new page allows the index to be easily enlarged when documents are added to the collection, and is more representative of larger collections where each query term is likely to require a page access. Hierarchy nodes that are siblings are stored together beginning on a new page; they take as few pages as possible without splitting a node across page boundaries. The vectors associated with sibling nodes are also packed into as few pages as possible. Having only siblings on a page (as opposed to packing hierarchy nodes as tightly as possible) allows the hierarchy to be easily updated when documents are added to or removed from the collection. It also makes the model more realistic of larger collections since this implementation renders back-tracking more expensive than it would be with a packed representation.

4.3. Processing Time Experiments

The efficiency statistics for the searches are presented in Table 3. The statistics given are the mean number of idealized machine instructions performed and the mean number of pages accessed for the

Table 3: Efficiency Statistics for Inverted Index and Cluster Searches

collection	inverted index search			cluster search		
	mean instructions	mean # pages	mean seconds	mean instructions	mean # pages	mean seconds
CACM	9,296	20.3	.62	129,230	155.1	4.78
CISI	8,339	15.1	.46	105,583	158.2	4.85
INSPEC	67,877	48.9	1.54	517,999	418.3	13.07
MED	3,136	18.1	.55	71,632	106.5	3.27

query set of each collection assuming each query is independent of all other queries and twenty documents are to be retrieved. Table 3 also includes the mean number of seconds to process a query assuming a page access takes .03 seconds (approximately 30 pages per second) and a machine instruction takes 1 microsecond.

The inverted index search is clearly more efficient than the cluster search: the cluster search performs 663% (INSPEC) to 2184% (MED) more machine instructions and accesses 488% (MED) to 948% (CISI) more pages than the inverted index search. The difference is caused by the fact that the inverted index search takes full advantage of the sparsity of the document vectors. In the inverted index search, only the terms that appear in *both* the document (or centroid) and the query are used. In the cluster search, a large percentage of the documents are avoided, but the centroids and documents that are encountered are exhaustively examined. Note in Figure 3(b) a term that occurs in only one of the query or document vector causes three machine operations but contributes nothing to the similarity; terms that occur in both vectors cause only six machine instructions. This problem is exacerbated in the search of the complete link hierarchy since that hierarchy is so bushy – every search begins by examining many long centroids.

5. Shorter Centroids

Both the extra disk space and the extra processing time needed by the cluster search have been attributed to centroids. Clearly if shorter centroids were used the efficiency of the search would improve, although the effectiveness might suffer. This section investigates the effect of decreasing the maximum length of centroid vectors.

5.1. Effectiveness

The cluster search was repeated on each collection using centroids of both maximum length 100 and maximum length 75. The effectiveness of the searches is summarized in Table 4. Three different statistics are given for each search: the mean precision after ten (top number) and twenty (bottom number)

Table 4: Effectiveness of Inverted Index and Cluster Searches

collection	inverted index search			cluster search centroids of length 75			cluster search centroids of length 100			cluster search centroids of length 250		
	mean prec	total # rel. ret.	# q's	mean prec	total # rel. ret.	# q's	mean prec	total # rel. ret.	# q's	mean prec	total # rel. ret.	# q's
CACM	.230 .185	115 185	9 5	.260 .215	130 215	8 7	.252 .213	126 213	9 7	.248 .199	124 199	8 6
CISI	.269 .260	94 182	7 5	.291 .241	102 169	9 6	.309 .253	108 177	7 3	.274 .237	96 166	6 3
INSPEC	.384 .315	296 485	3 2	.387 .308	298 475	10 5	.397 .314	306 484	7 4	.373 .284	287 437	6 4
MED	.620 .528	186 317	0 0	.660 .600	198 360	0 0	.667 .600	200 360	0 0	.650 .603	195 362	0 0

documents are retrieved, the total number of relevant documents retrieved by all queries after ten and twenty documents are retrieved, and the total number of queries that retrieved no relevant documents after ten and twenty documents are retrieved.

For all centroid lengths, the cluster search is more effective than the inverted index search after ten documents are retrieved; in fact, the mean precision of the cluster search using centroids of maximum length 100 is greater than that of the search using centroids of maximum length 250. Note, however, that more queries retrieved no relevant documents in the search that used centroids of maximum length 100. In general, the search that used centroids of maximum length 75 had more queries that retrieved no relevant documents than any other search, and its mean precision was worse than that of the search in which centroids were restricted to a maximum length of 100.

5.2. Disk Space

The disk space in bytes used by the cluster search is given in Table 5; also included in the Table are the percentage decrease in the size of shorter centroids to the size of longer centroids, and the percentage increase of the total size for the cluster search to the total size for the inverted index search. Although there is some decrease in the required space when the centroids are shortened, the decrease is not large enough to make any centroid file smaller than the inverted file of the document collection (see Table 2). For the large INSPEC collection, the total size needed by the cluster search with centroids of maximum length 75 is still 47% larger than the total size needed by the inverted index search.

Note that the space required by the centroids does not decrease as much as might be expected when the maximum centroid length is reduced to 100 from 250. Indeed, a decrease of at most 37% is observed in the size of the centroid file (whereas the maximum centroid length was reduced by more than half). This

Table 5: Cluster Search Sizes for Varying Maximum Centroid Lengths

	centroid file size in bytes			% decrease in centroid size			% increase in total size over inverted index search		
	75	100	250	100/250	75/250	75/100	75	100	250
CACM	978,376	1,136,896	1,481,320	23	34	14	34	47	74
CISI	766,328	936,240	1,375,536	32	44	18	23	39	73
INSPEC	6,110.384	7,160,584	9,685,704	26	37	15	47	62	99
MED	564,864	711,512	1,132,544	37	50	21	11	26	71

minimum amount of reduction can be attributed to the shape of the complete link hierarchy: since the hierarchy contains small, tight clusters, some clusters do not contain more than 100 distinct terms. More clusters contain at least 75 terms, so the decrease in size from a maximum length of 100 to 75 terms is proportionally larger.

5.3. Processing Time

Although some clusters do not contain 100 terms, top-level clusters are the most likely clusters to contain many terms since they are the largest clusters in the hierarchy. Thus in spite of the relatively small reduction in size, significant processing time may be saved by using shorter centroids in the cluster search.

The processing time results are given in Table 6. The inverted index search is still the most time efficient search. Unsurprisingly, using shorter centroids is always more efficient than using longer centroids. Somewhat more surprisingly, only the INSPEC collection exhibits a large decrease in the number of pages accessed. An explanation for this behavior can be found in Table 7 which contains the mean number of pages accessed to retrieve hierarchy nodes and the mean number of pages accessed to retrieve vectors. When shorter vectors are used, the number of pages accessed to retrieve vectors decreases but the number of pages accessed to retrieve hierarchy nodes increases. It is harder to distinguish among clusters when shorter centroids are used, and thus the search with shorter centroids is broader than the search with longer centroids. The slight increase in the number of pages accessed due to backtracking is negligible only for the large INSPEC collection.

6. Summary

In earlier work, a top-down search of the complete link hierarchy was shown to be more effective than an inverted index search [GRIF84, VOOR86]. Unfortunately, the results of this study show the cluster search is much more expensive than the inverted index search in terms of both disk space and processing time. Decreasing the maximum length of centroids increases the efficiency of the cluster search,

Table 6: Processing Time of Inverted Index and Cluster Searches

collection	inverted index search			cluster search centroids of length 75			cluster search centroids of length 100			cluster search centroids of length 250		
	mean instrs	mean pages	secs	mean instrs	mean pages	secs	mean instrs	mean pages	secs	mean instrs	mean pages	secs
CACM	9,296	20.3	0.62	78,524	133.6	4.09	91,186	138.7	4.25	129,230	155.1	4.78
CISI	8,339	15.1	0.46	50,568	148.1	4.49	61,021	147.3	4.48	105,583	158.2	4.85
INSPEC	67,877	48.9	1.54	221,141	218.1	6.76	276,867	257.2	7.99	517,999	418.3	13.07
MED	3,136	18.1	0.55	32,109	87.1	2.65	39,581	90.9	2.77	71,632	106.5	3.27

especially for large collections, and the centroids can be made quite short before the effectiveness of the cluster search suffers. However, the centroid length cannot be reduced enough to make the cluster search as efficient as the inverted index search while remaining as effective as the inverted index search.

It should be stressed that these experiments try only one method of increasing the efficiency of the cluster searches. Other ways of increasing the efficiency of the cluster search without sacrificing its effectiveness should be investigated. For example, inverting the top-level centroids of the hierarchy and starting the cluster search by doing an inverted search on those centroids should save time and would not increase the disk storage (since the inverted centroids could be removed from the direct centroid file). Furthermore, the effectiveness would remain the same since the same clusters would be explored.

Similarly, it may be possible to modify the complete link hierarchy to make searching it easier. An obvious possibility is to cluster the top-level centroids using some other clustering method and thus add several new levels to the hierarchy. Such a hybrid hierarchy would be more time efficient to search since fewer centroids would be at the new top level, but the same effectiveness would no longer be guaranteed.

Table 7: Mean Page Accesses to Retrieve Hierarchy Nodes and Vectors in Cluster Search

collection	centroids of length 75		centroids of length 100		centroids of length 250	
	hierarchy	vectors	hierarchy	vectors	hierarchy	vectors
CACM	49.0	84.3	47.4	91.4	43.6	114.7
CISI	66.6	81.6	63.2	84.2	56.7	105.7
INSPEC	53.0	165.0	54.6	202.6	41.5	354.5
MED	38.5	48.5	38.5	52.5	35.3	69.3

Cluster searching also affords the possibility of storing the natural language text of a document with its representative in the cluster hierarchy. This may increase the efficiency of the cluster search by avoiding additional page accesses when displaying the retrieved documents to the user. This enhancement is not possible with the inverted index search since the terms of a document are usually spread throughout the index.

Acknowledgments

Thanks go to Professor Gerard Salton and the other members of the SMART group at Cornell for many helpful discussions, and especially for suggestions on how to improve the efficiency of the cluster searches.

References

[BUCK85] Buckley, C. A., Lewit, A. F., "Optimization of Inverted Vector Searches", *Proceedings of the Eighth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 97-110, 1985. ACM order number 606850.

[FOX83] Fox, E. A., "Characterization of Two New Experimental Collections in Computer and Information Science Containing Textual and Bibliographic Concepts", Technical report 83-560, Department of Computer Science, Cornell University, September 1983.

[GRIF84] Griffiths, A., Robinson, L. A., Willett, P., "Hierarchic Agglomerative Clustering Methods for Automatic Document Classification", *The Journal of Documentation* XL(3):175-205, 1984.

[GRIF85] Griffiths, A., Luckhurst, H. C., Willett, P., "Using Interdocument Similarity Information in Document Retrieval Systems", *Journal of the American Society for Information Science* XXXVII(1):3-11, 1986.

[JARD71] Jardine, N., van Rijsbergen, C. J., "The Use of Hierarchic Clustering in Information Retrieval", *Information Storage and Retrieval* VII(5):217-240, 1971.

[SMAR71] Salton, G., editor, *The SMART Retrieval System: Experiments in Automatic Document Processing*. Prentice-Hall, 1971.

[VOOR86] Voorhees, E. M., *The Effectiveness and Efficiency of Agglomerative Hierarchic Clustering in Document Retrieval*, PhD thesis, Cornell University, 1986.