The Effectiveness and Efficiency of Agglomerative Hierarchic Clustering in Document Retrieval

Ellen M. Voorhees
Ph.D. Thesis

TR 85-705
October 1985

Department of Computer Science
Cornell University
Ithaca, NY 14853

The Effectiveness and Efficiency of Agglomerative Hierarchic Clustering in Document Retrieval

Ellen Marie Voorhees, Ph.D.

Cornell University 1986

The major component of a document retrieval system is the component that searches the document collection and selects the documents to be returned in response to a query. Since users wait for the results of the search, the component must be efficient as well as effective. The main goal of this thesis is to compare clustered file searches and inverted file searches in order to determine under what circumstances one search is to be preferred over the other. A preliminary goal is to define a good cluster search.

Three types of agglomerative clustering strategies, the single link, the complete link, and the group average link methods, are investigated. Searches of the single link hierarchy, the cluster hierarchy used extensively in previous research, are shown to be inferior to searches of the other hierarchy types. Searches of the group average link and complete link hierarchies perform similarly for small collections; for larger collections, searches of the complete link hierarchy are more effective. A top-down search of the group average link hierarchy is the most time efficient search asymptotically.

The experimental evidence suggests that the difference in the efficiency and effectiveness of the complete link and group average link searches is due to the restricted depth of the complete link hierarchy. The depth of the group

average link hierarchy increases as the size of the collection increases, but the depth of the complete link hierarchy does not. Thus the largest clusters in the complete link hierarchy are not very large, and the clusters can be accurately represented by centroids. Since the depth of the hierarchy does not increase with collection size, searches of the complete link hierarchy should remain effective for larger collections.

The top-down search of the complete link hierarchy is somewhat more effective than the inverted file search. The relative efficiency of the two searches depends on the relative efficiency of accessing a page and computing a similarity, since the cluster search accesses many more pages but computes fewer similarities than the inverted file search. For an inexpensive similarity measure, the inverted file search is much more efficient.

THE EFFECTIVENESS AND EFFICIENCY OF AGGLOMERATIVE HIERARCHIC CLUSTERING IN DOCUMENT RETRIEVAL

A Thesis
Presented to the Faculty of the Graduate School
of Cornell University
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy

by
Ellen Marie Voorhees

January 1986

Biographical Sketch

Ellen Marie Voorhees was born in Bensalem, Pennsylvania on March 13, 1958. She attended Bensalem Senior High School, from which she graduated as valedictorian in June of 1976. She was graduated with highest distinction from the Pennsylvania State University in 1979 after earning the Bachelor of Science degree in computer science. In 1982 she earned the Master of Science degree in computer science from Cornell University. She is a member of Phi Kappa Phi and the Association for Computing Machinery.

Acknowledgments

My sincere thanks to my thesis advisor Gerard Salton. His insightful comments have improved both the content and the style of this thesis immeasurably. I also wish to thank Robert Constable and Lee Schruben for serving on my committee.

I have had the benefit of many conversations with the people affiliated with the SMART project in the past several years: Chris Buckley, Carolyn Crouch, Donald Crouch, Joel Fagan, Ed Fox, Maria Smith, and Yuankai Zhang. Most of the ideas included in this thesis were developed in seminars with these people. Geri Pinkham, project secretary, has always been extremely helpful.

The inclusion of the group average link and complete link hierarchies in this research was motivated by the work of Peter Willett and his colleagues at the University of Sheffield. He has been very generous in sending me preprints of his current research and telling me about other relevant research.

I gratefully acknowledge the support and enthusiasm of my family and friends. Special thanks to Jim Sasaki who learned far more than he ever wanted to know about document clustering over lunch.

Finally, my warmest thanks to Christopher. As my husband and colleague he was always the first to hear my ideas and read my drafts. His confidence and encouragement have enabled me to complete this thesis.

Table of Contents

List of Tables

List of Figures

Chapter 1
Introduction

The goal of an *information retrieval system* is to efficiently retrieve from a body of documents all of the documents, and only those documents, that meet the user's information need. Here a *document* is any unit of text that is of interest; common examples of documents are abstracts of scientific articles, electronic messages, and chapters of books. The information need of the user is expressed in a *query*. The query is matched with document representatives, and those documents that meet the selection criteria are returned to the user. This overview of the operation of an information retrieval system is necessarily vague. Since no system is presently capable of obtaining the goal of efficient, perfect retrieval, different models of the retrieval process – different ways of representing documents and queries, different ways of matching the query and document representatives, different selection criteria – are in use. The assumptions on which these various models are based reflect different efficiency and effectiveness compromises.

Most commercial retrieval services use systems based on the *Boolean model.* In this model each document is represented by a list of terms. A query is a Boolean expression of terms that appear in documents. The matching function and selection criteria are the same: every document that satisfies the query (*i.e.* every document whose term assignment causes the Boolean expression to evaluate to true) is retrieved. Usually a Boolean system is implemented using an *inverted index* of the document collection. An inverted index is a set of lists of document identifiers, one list for each term in the collection. The list for a given term contains the identifiers of all the documents in which the term appears. The set of documents that satisfy the query can then be found by simple list manipulations of the inverted lists corresponding to the query terms.

Boolean systems can be efficiently implemented. However, they exhibit several disadvantages [Salton 75, pp. 121,123]:

- although in principle it is possible to create a Boolean expression that exactly specifies an information need, in practice formulating good Boolean queries is a difficult art;
- the only mechanism that controls the number of documents that are returned is the specificity of the query;
- neither the terms in the documents nor the queries are weighted to reflect their relative importance;
- the documents that are returned are not ranked in presumed order of importance to the user.

To overcome these disadvantages of Boolean systems, some retrieval systems are based on the *vector space model.* In this model documents and queries are represented by vectors in T-dimensional space where T is the number of distinct terms in the document collection. The matching function is a measure of the similarity (or distance) between the document and query vectors. For example, the SMART system [Salton 71a] has used the cosine of the angle between the vectors as the similarity measure. The documents are ranked in decreasing order of their similarity to the query, and the most similar documents are returned.

For efficiency reasons, it is infeasible to compute the similarity of each document with the query by directly accessing each document. One method of avoiding this *sequential scan* of the document collection is to compute only the non-zero similarities by using an inverted index to identify the documents that have at least one term in common with the query [Smeaton & van Rijsbergen 81]. A second method is to structure the collection by placing documents into groups based on how similar they are to one another. The search can then be limited to those groups of documents that are likely to contain the documents that are the most similar to the query [Salton 71a, Part IV]. Forming the groups of similar documents is called *document clustering*; searching the resulting clusters is called *cluster searching*.

The inverted index search computes all non-zero query-document similarities, so the ranking produced by that search will be identical to the ranking produced by a sequential scan of the document collection. A cluster search, however, does not compute all non-zero query-document similarities and may take into account some document-document similarities. Thus the ranking produced by a cluster search is likely to differ from that produced by a sequential scan. Widely varying claims have been made about the relative effectiveness and efficiency of these two searching strategies. Murray compared cluster searching with sequential searching and concluded that although the cluster search suffered a loss in effectiveness, it was more efficient in terms of both external memory requirements and the number of disk accesses required to perform a search [Murray 72]. Jardine and van Rijsbergen [Jardine & van Rijsbergen 71] and more recently Griffiths, Robinson, and

Willett [Griffiths, Robinson & Willett 84] have claimed other types of cluster searches could be more effective than a full ranking of the document collection.

This thesis is a response to these conflicting conclusions. Search and selection strategies are the major components of a retrieval system and are the components which must operate in real time. If the documents that are selected are not helpful, or if they are helpful but are not returned to the user before he or she grows impatient, the retrieval system will not be used. Thus the main purpose of this thesis is to compare the effectiveness and efficiency of inverted index and cluster searches in order to determine the circumstances in which one strategy is to be preferred over the other. In the course of making these comparisons different clustering methods and cluster searching strategies will also be contrasted.

1.1. Background

This section reviews previous document clustering studies. It is not a comprehensive review of the document clustering literature, but rather provides the appropriate background for the current research. The following section provides a more complete description of the thesis.

1.1.1. SMART Clustering Experiments

Early clustering work performed by the SMART project used document clustering solely as a way to avoid computing the similarity between the query and each document. Documents were grouped into classes and each class was represented by a single vector known as the *centroid* vector. To perform a search, the similarity between the query and each centroid vector was computed, and only the documents in clusters whose centroid was similar enough to the query were examined. If there were many clusters, the process was repeated – centroid vectors were grouped together and represented by supercentroids, supercentroids were grouped together and represented by hypercentroids, *etc.* – forming a hierarchic index into the collection. The search on each level of the hierarchy identified the portion of the next lower level that needed to be searched. (Searches that start at the top of the hierarchy and proceed towards the documents at the bottom are called *top-down* searches. *Bottom-up* searches, on the other hand, begin near the document level and proceed towards the top.)

Various methods for creating the document clusters were used. Rocchio [Rocchio 66] developed a method in which documents that are located in sufficiently dense areas of the vector space become cluster seeds, and the remaining documents are then allocated into the most similar cluster(s). Dattola [Dattola 71] extended a method by Doyle [Doyle 66] that iteratively refines an (arbitrary) initial set of clusters. Rieber and Marathe [Rieber & Marathe 69] introduced a method that creates a set of clusters in a single scan of the document collection. Williamson [Williamson 74] devised an algorithm that inserts documents into a pre-existing hierarchy by treating them as queries. Other methods clustered documents by first clustering queries previously submitted to the system and assigning documents to clusters based on their relationship to queries in the same clusters [Lesser 66; Worona 71; Yu 73].

Each of these clustering algorithms has an attractive running time. Most of them can cluster a collection of N documents in at most $O(N \log N)$ time. Unfortunately, these algorithms are defined heuristically and depend on the values of a variety of parameters that control such hierarchy characteristics as the number of clusters formed, the maximum amount of overlap allowed between clusters, and the minimum and maximum sizes of clusters. Appropriate values for these parameters are collection dependent and must be determined experimentally. Furthermore, the final classification depends on the order in which the documents are added to the clustered collection or on the initial set of clusters.

Experiments were performed on all the hierarchy types mentioned above using searches that differed in the number of nodes that were expanded and the amount of backtracking allowed. The following conclusions were drawn:

- Expanding more than one cluster on each level produced recall-oriented searches while expanding only one cluster produced precision-oriented searches* [Salton & Wong 78].

- The precision of the clustered searches at a given recall level was less than the precision of the full search. The difference was small in the middle recall range, but pronounced at high recall levels [Salton 71b].

- Searches of hierarchies that had many small clusters were more effective but less efficient than searches of hierarchies that had fewer, larger clusters [Grauer & Messier 71].

- Cluster searches were much more efficient than a sequential scan of the document collection [Salton 71b].

---

* Recall is the proportion of relevant documents that are retrieved; precision is the proportion of retrieved documents that are relevant

1.1.2. Cluster Based Retrieval

Jardine and van Rijsbergen also used top-down searches of hierarchically clustered collections [Jardine & van Rijsbergen 71], but their work differed from the SMART work in at least three fundamental ways. First, the purpose of the cluster searches was different. Jardine and van Rijsbergen thought cluster searching could not only be more efficient than a sequential scan of the collection, but could also be more effective than a sequential scan. Their reasoning was based on an assumption called the *cluster hypothesis*: "the associations between documents convey information about the relevance of documents to requests", or more simply, documents that are similar to one another tend to be relevant to the same queries.

Secondly, Jardine and van Rijsbergen used a selection mechanism they called *cluster based retrieval*. Cluster based retrieval returns entire cluster(s) in response to a query in contrast to the SMART method of ranking the documents within the selected clusters by decreasing similarity to the query. The rationale for this selection mechanism is the cluster hypothesis. The cluster hypothesis implies the document-document similarities used to construct the cluster provide important information about the relevance of documents within the cluster to the query. One way of incorporating this information into a retrieval strategy is to select entire clusters for retrieval instead of selecting documents from within clusters on the basis of the individual documents' similarity to the query.

Finally, the hierarchies used by Jardine and van Rijsbergen and the SMART group were very different. Jardine and van Rijsbergen used single link clustering. The formal definition of the single link hierarchy will be given in Chapter 2. For now, only the differences between the single link and the SMART hierarchies are important. One important difference is that the single link hierarchy has a formal definition and therefore its creation is independent of experimentally defined parameters. Other differences may be noticed by considering the hierarchies in Figures 1.1 and 1.2.

In Figures 1.1 and 1.2 the hierarchies are represented by trees in which the leaves represent documents and the interior nodes represent non-singleton clusters. All the documents that are descendants of a node belong to the cluster represented by that node. For example, the root of the tree represents the cluster consisting of the entire document collection. The SMART hierarchy, Figure 1.1, is balanced and has all the documents on the same level. The *low-level clusters*, the set of clusters such that each cluster is the smallest cluster to contain some document, are approximately the same size. Clusters may overlap as illustrated by document B in Figure 1.1. In the single link hierarchy, Figure 1.2, the documents may appear at any level. The tree is deep and narrow and the sizes of the low-level clusters range from two to the size of the collection. Clusters "overlap" only in the sense that smaller clusters are nested within larger clusters.

The effectiveness of cluster based retrieval on single link hierarchies was examined through both *retrospective* and *predictive* retrieval experiments [Jardine & van Rijsbergen 71; van Rijsbergen 74; van Rijsbergen & Croft 75; Croft 78]. In retrospective experiments full relevance information is assumed to be known at the time of the search, and retrieval decisions are made to optimize performance. Predictive searches model the behavior of operational

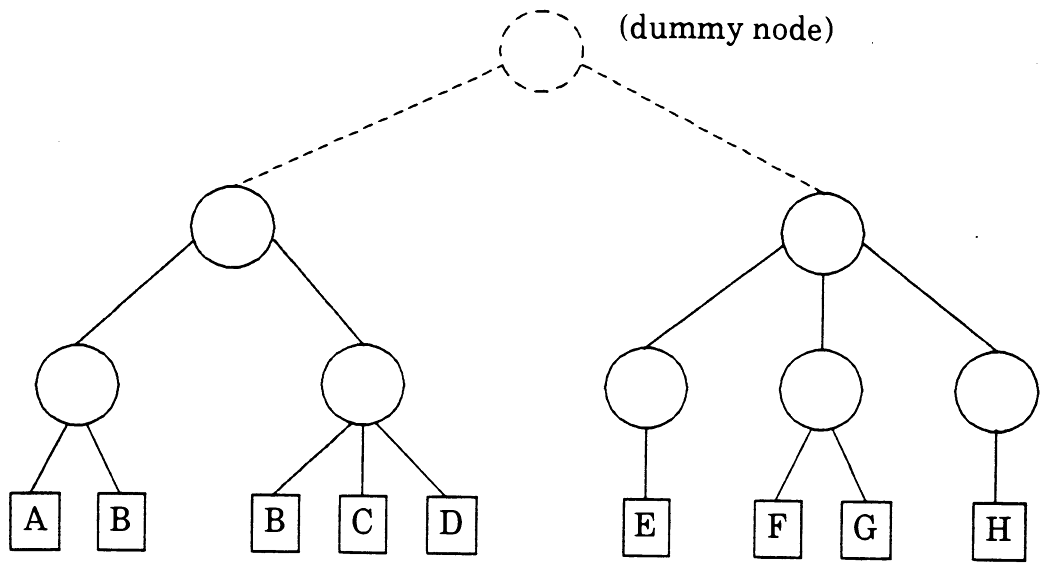


Figure 1.1: A sample SMART hierarchy with overlap
(adapted from [Murray 72])

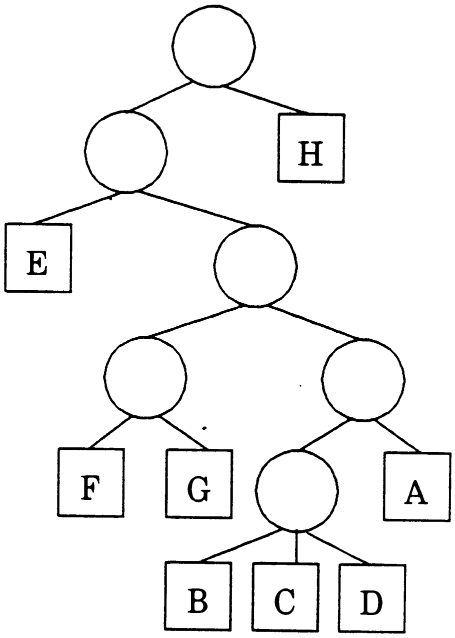


Figure 1.2: A sample single link hierarchy

retrieval services where relevance information is not known at the time of the search.

The retrospective experiments compared the effectiveness of retrieving the best cluster with the effectiveness of retrieving the top n documents from a full ranking. (The value of n that optimized the average performance over all queries was used.) For each of the collections tested, the optimal performance of cluster based retrieval was better than the optimal performance of the sequential scan, especially for precision-oriented searches [Jardine & van Rijsbergen 71; van Rijsbergen 74; van Rijsbergen & Croft 75].

In the predictive experiments a variety of different searching strategies were compared with the aim of determining a strategy that closely approximated the ideal of always retrieving the optimal cluster. The searches differed in which cluster(s) were retrieved. Entire clusters were retrieved in all cases. (In the description of the searches that follows, reference will be made to the similarity between a cluster and a query or between a [hierarchy] node and a query. These phrases are a shorthand for the more precise "similarity between the centroid of a cluster and a query" or "similarity between the centroid of the cluster represented by a hierarchy node and a query".)

Three different types of top-down searches were performed. The first type, the simplest, was a straightforward depth-first search of the hierarchy [Jardine & van Rijsbergen 71]. The search started at the top of the hierarchy and the similarity between the query and each of the children of the root was computed. The child with the largest similarity was selected and the query-centroid similarities of all the non-leaf children of that node were computed.

The process was repeated until either all the similarities between the query and the non-leaf children of some node, x, were less than the similarity between the query and x, or all the children of x were leaves (documents). The cluster represented by x was retrieved.

The second top-down searching method was broader than the one described above in that two branches were explored on each level [van Rijsbergen & Croft 75]. However, only one cluster was retrieved for each query.

The third method was broader still [van Rijsbergen 74]. Each path down the tree was started, but a path was abandoned once the similarities failed to meet certain criteria. Experimental evidence suggested that almost all paths were abandoned very early in the search, preserving the efficiency of the search. All the clusters that satisfied the criteria were ranked by decreasing similarity to the query, and the best half were retrieved in their entirety.

A bottom-up search was also used [van Rijsbergen & Croft 75]. Access into the hierarchy at the bottom level was obtained through an inverted index of the low-level centroid terms. The inverted index of centroid terms was used to find the low-level cluster that was most similar to the query. The search then continued up the tree until the similarity between the query and the parent of the current node was smaller than the similarity between the query and the current node. The cluster represented by the current node was retrieved.

Although none of the searches performed nearly as well as the optimal cluster search, some of the precision-oriented cluster searches were more effective than the full ranking with a single (optimal) cut-off. The broad top-down search was always more effective than the narrow depth-first search, and, when recall was more important than precision, it was also more effective than the bottom-up search. The bottom-up search was more effective than any of the top-down searches when precision was emphasized. These results are in keeping with the probabilistic model of cluster based retrieval developed by Croft [Croft 80]. The results of experiments based on that model indicated the narrow top-down search of the single link hierarchy was more likely to select the wrong cluster than the bottom-up search of that hierarchy was.

Griffiths and Willett compared cluster based retrieval on the single link hierarchy and hierarchies produced by similar clustering methods, namely the complete link, the group average link, and the Ward methods [Griffiths & Willett 84; Griffiths, Robinson & Willett 84]. Once again both retrospective and predictive searches were performed. Although few significant differences were found in the effectiveness of the predictive searches, the retrospective experiments showed that the optimal effectiveness obtainable by retrieving a single cluster from the single link hierarchy was significantly worse than the optimal performance obtainable by retrieving a single cluster from the hierarchies produced by the other clustering methods.

1.2. Thesis Synopsis

This thesis has two goals. The first goal is to systematically compare the different clustering methods and cluster search strategies that have been introduced in earlier work in order to determine the most effective and efficient cluster searches. The second goal is to compare the efficiency and effectiveness of suitable cluster searches to that of the inverted index search in order to determine when one search is to be preferred over the other.

The three clustering methods to be investigated are the *single link*, the *complete link*, and the *group average link* methods. Each of the methods is an instance of a general class of clustering methods known as *agglomerative hierarchic* methods. In such clustering methods, the items to be clustered are first considered to belong to the cluster consisting of the item itself. The two closest clusters are successively merged until only one cluster remains. The definition of closest depends on the clustering method being used. Agglomerative hierarchic methods were chosen because the theoretical properties of these methods are well understood [Lance & Williams 67; Jardine & Sibson 68]. Single link, complete link, and group average link were chosen since they include the most well known of the agglomerative hierarchic methods and since they have very different clustering properties.

Two different searching strategies, bottom-up and top-down, and two different selection mechanisms, retrieving clusters in their entirety and retrieving individual documents from within clusters, are explored. Although Croft's work suggests that bottom-up searching is expected to be more effective than top-down searching [Croft 80], top-down searches are also investigated since Croft worked only with single link clustering and retrieved only very few (approximately two) documents per query on average. Also, a narrow top-down search is likely to be more efficient than a bottom-up search. Both retrieving entire clusters and retrieving individual documents from within clusters are investigated since no direct comparisons between these two selection mechanisms has been made. Croft and van Rijsbergen compared their results to Murray's work [van Rijsbergen and Croft 75], but the differences that were found could as easily have been the result of the (very) different hierarchies in use as of the different selection mechanisms.

The next chapter describes the retrieval environment used in this thesis. The p-norm model [Salton, Wu & Fox 83] which includes the Boolean and vector space models as special cases is described. The formal definition of the clustering methods, as well as a discussion of their theoretical properties and of the algorithms used to construct the hierarchies follow. The different cluster search algorithms and the inverted index search are described in detail.

Chapter 2 concludes with a description of the document collections used in all the experiments. Four collections, ranging in size from 1033 to 12,684 documents, are used. The description of the collections includes the results of a new test for the cluster hypothesis which shows that the hypothesis characterizes the collections to varying extents. The use of several collections with different characteristics assures the conclusions reached in this thesis are not dependent on some peculiarity of a test collection.

Chapter 3 explores the effectiveness of cluster searches in comparison to each other and to the effectiveness of ranking the entire document collection by decreasing similarity to the query. Three main conclusions are drawn from these experiments.

- Cluster searching can be more effective than ranking the entire document collection by decreasing similarity to the query.

- Searches of the complete link hierarchy are usually more effective than searches of the group average link hierarchy which in turn are more effective than the searches of the single link hierarchy. The evidence suggests that the effectiveness of the complete link method is due to the shallow, bushy hierarchy created by that clustering method.

- Retrieving individual documents from within clusters is usually more effective than retrieving clusters in their entirety.

Chapter 4 discusses the efficiency of the searches in terms of both CPU processing time and I/O time. The inverted index search time is shown to be dominated by the CPU factor while the cluster search time is dominated by the I/O factor. A study of how the total search times compare when different similarity measures (with different costs) are used is included in the chapter. The effective cluster searches are shown to be much less efficient than the inverted index search when an efficient similarity measure is used. For similarity measures such as the p-norm measure that are more expensive to compute, the total processing time of the cluster search is smaller than the total processing time of the inverted index search for collections greater than a certain size. (Precisely what size depends on just how expensive the similarity measure is and on other similar factors.) Since the processing time of the inverted index search is proportional to the length of the query, the efficiency experiments are repeated using longer feedback queries. The results of these experiments suggest that the relative amounts of processing time for the cluster and inverted index feedback searches remains approximately equal while the differences in effectiveness narrow.

The final chapter reviews the effectiveness and efficiency results and makes recommendations for operational retrieval services. Suggestions for future work are included at the end of the chapter.

Chapter 2
The Retrieval Environment

Although the purpose of this thesis is to investigate search and selection strategies, the other components of the retrieval process – document and query representation, selection of index terms, term weighting, query-document similarity computations – necessarily affect the performance of the searches, and, therefore, the choices made in these areas need to be discussed in addition to the specific algorithms used for clustering and searching. Accordingly, this chapter describes the retrieval environment in which the experiments described in this thesis take place. All the experiments are performed using the current version of the SMART information retrieval system [Buckley 85] extended by programs specifically designed for this work.

The first section provides an introduction to the p-norm model of information retrieval. The next section defines the three clustering methods and discusses their theoretical properties. Implementation details are included in that section. The third section presents the inverted index and cluster search algorithms. The final two sections cover topics that are peculiar to experimental information retrieval systems: evaluation of retrieval performance for the purpose of comparing different strategies and the description of test collections including the properties of relevant documents.

2.1. The P-norm Model

The *p-norm model* of information retrieval [Salton, Wu & Fox 83] is an extended Boolean model based on the ℓ_p family of norms. The p-norm retrieval method is based on a strategy similar to that used in vector processing except

$$sim(<A,a>,D) = aD_A$$

$$sim([\textbf{not } t],D) = 1 - sim(t,D)$$

$$sim([<t_1,w_1> \textbf{or}^p <t_2,w_2> \textbf{or}^p \dots \textbf{or}^p <t_n,w_n>],D) =$$

$$\left[\frac{w_1{}^p sim(t_1,D)^p + w_2{}^p sim(t_2,D)^p + \dots + w_n{}^p sim(t_n,D)^p}{w_1{}^p + w_2{}^p + \dots + w_n{}^p} \right]^{1/p}.$$

$$sim([<t_1,w_1> \textbf{and}^p <t_2,w_2> \textbf{and}^p \dots \textbf{and}^p <t_n,w_n>],D) =$$

$$1 - \left[\frac{w_1{}^p(1\text{-}sim(t_1,D))^p + w_2{}^p(1\text{-}sim(t_2,D))^p + \dots + w_n{}^p(1\text{-}sim(t_n,D))^p}{w_1{}^p + w_2{}^p + \dots + w_n{}^p} \right]^{1/p}$$

Figure 2.1: Definition of p-norm similarities

that quasi-Boolean queries are used. In particular, a similarity measure is computed between a query and the documents identified by sets of weighted terms. This allows the documents to be ranked by decreasing order of similarity. Both query and document terms may be weighted; the interpretation of the Boolean operators is controlled by a parameter known as the *p-value.*

2.1.1. Definition of the Model

The similarity between a document vector and a p-norm query is defined recursively using the equations in Figure 2.1. In the figure, D represents the

the document vector and D_A represents the weight of term A in the document. An arbitrary p-norm clause is represented by t_i and its weight by w_i, $0 \leqq w_i \leqq 1$.

For example, if Q = {<A,a> **or** ([<E,e> **and** <F,f>],b)}, then

$$\mathrm{sim}(Q,D) = \left[\frac{a^p d_A + b^p \left[1 - \left[\dfrac{e^p(1-d_E)^p + f^p(1-d_F)^p}{e^p + f^p} \right]^{1/p} \right]^p}{a^p + b^p} \right]^{1/p}.$$

Note that clauses as well as concepts may be assigned a weight (b is the weight of the clause (<E,e> **and** <F,f>) in the example above).

The value of the parameter p determines the strictness of interpretation of the Boolean **and** and **or** operators [Salton, Wu & Fox 83; Wu 81]:

a) When p is equal to infinity, and binary weights are used, the result is a conventional Boolean system. The similarity measurements provide values of one for documents that match the queries and hence are to be retrieved, and values of zero for the remainder that needs to be rejected.

b) As the value of p is reduced from infinity, the interpretation of the Boolean operators is less and less strict. A clause such as (A **and** B) is then treated as a tentative phrase, rather than a compulsory one, in the sense that a document exhibiting both items still receives a perfect score; however a document exhibiting one of the two terms now receives a partial sum rather than a null score as in the conventional system. Analogously, in a clause such as (A **or** B), the two terms are treated as approximately, rather than completely, synonymous. This implies that a document that includes both terms will be preferred over one containing only one of the terms.

c) As p reaches its lower boundary of one, the formulas for **and** and **or** queries produce the same result. In that case, the presence of two query terms in a document is worth twice as much as the presence of one term (assuming fully weighted terms), and the distinctions between **and** and **or** are lost. When p is equal to one, the extended Boolean system is reduced to a vector processing system, and the Boolean queries (A **and** B) and (A **or** B) are then equivalent to the vector query (A,B) specifying two terms without special structural information.

2.1.2. Indexing in the P-norm Model

The creation of the document vectors from natural language text is done automatically. The text of a document is scanned to identify words, and words found on a standard stop list are removed. These words are function words such as prepositions that contain no content information. Word variants are mapped into a common *stem* and the stem is assigned an unique *concept number*. The vector consists of a set of concept number, weight pairs.

The weights are also assigned automatically. The particular weights used in this work are normalized term frequency times inverse document frequency weights as suggested by Fox [Fox 83a]. A *term frequency* (tf) weight for a term in a given document is proportional to the number of times the term appears in the document. An *inverse document frequency* (idf) weight of the term is inversely proportional to the number of documents in which the term appears. If

tf_i is the number of times term i appears in a given document; idf_i is the inverse document frequency of term i, specifically the natural log of the maximum number of documents in which any term appears divided by the number of documents in which this term appears; and tf_{max} is the maximum number of times any term appears in the given document, then the weight of term i in the given document is computed as

$$idf_i \times \left[.5 + .5 \frac{tf_i}{tf_{max}} \right]. \tag{2.1}$$

For convenience in processing, these weights are further normalized so that the sum of the squares of the weights in each vector equals one.

P-norm queries may be automatically created from a natural language statement of the query [Salton, Buckley & Fox 83]. The queries used for these experiments were, however, manually constructed Boolean queries with term weights and p-values added automatically. The indexing process for queries is the same as that for documents with the exception that a query concept that appears in no document is dropped from the query. The weights of the concepts in the query are normalized inverse document frequency weights since the term frequency component is always one. (Even if a concept appears in the query more than once, its term frequency weight is still one since weights are assigned recursively by clauses. The base clause is a single concept which has a term frequency weight of one.) Specifically, the weight of term i in a query is

$$\ln \left[\frac{\text{collection size}}{\text{number of documents which contain term i}} \right] \div \ln \left[\frac{\text{collection size}}{1} \right].$$

Clause weights are the mean weight of the constituent sub-clauses. Previous experiments suggested that small p-values are beneficial [Salton, Wu & Fox 83; Fox 83a; Salton & Voorhees 85]. Accordingly, all p-values in these experiments are equal to two.

2.2. Agglomerative Hierarchic Clustering

Single link, complete link, and group average link clustering, the clustering methods used in this work, are members of the class of agglomerative hierarchic clustering methods. These methods depend on the similarities* between documents and not on the actual terms that make up the documents as in Litofsky's work [Litofsky 69]. When defining the clustering methods, it is convenient to think of the input to the clustering algorithms as an $N \times N$ similarity matrix where N items are to be clustered, although in practice the entire similarity matrix seldom needs to be computed. It is assumed that the similarity measure is symmetric (*i.e.* $sim(a,b) = sim(b,a)$).

2.2.1. Definitions

The general approach of each of the agglomerative methods is the same. Before any clustering has taken place, each document to be clustered forms a singleton cluster. The similarity between clusters is the similarity between the documents in the respective clusters. The two most similar clusters (ties are broken arbitrarily) are merged to create a new cluster consisting of all the

---

* I define the clustering methods in terms of similarities. However, equivalent definitions for distances are obtained by substituting "distance" for "similarity", and interchanging "minimum" and "maximum".

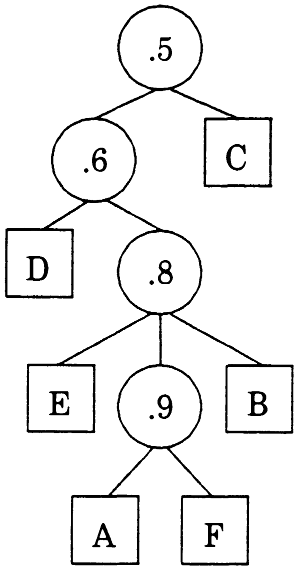
items in the original two clusters. The similarity between this new cluster and the remaining clusters is computed and the process is repeated. When only one cluster remains, the process stops.

The methods differ in how the similarity between non-singleton clusters is defined. In the single link method the similarity between two clusters is the maximum of the similarities between all pairs of documents such that one document of the pair is in one cluster and the other document is in the other cluster. The definition of the complete link method is just the opposite: the similarity between two clusters is the minimum of the similarities between all pairs of documents such that one document of the pair is in one cluster and the other document is in the other cluster. The similarity between two clusters in the group average link method is the mean of the similarities between all pairs of documents such that one document of the pair is in one cluster and the other document is in the other cluster. The hierarchies defined by each of these methods for a sample set of data are given in Figure 2.2. In the Figure the similarity level at which a cluster forms is recorded in the node that represents the cluster.

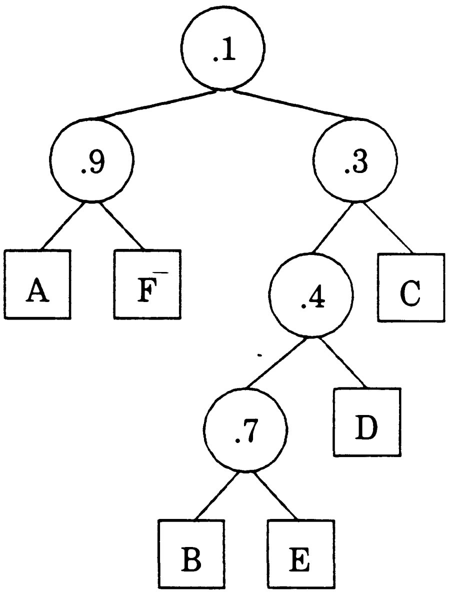
Although the differences among the hierarchy types are minimized when the collection consists of only six items, some differences can be observed in Figure 2.2. The single link clusters form at larger similarity values than either the complete link or group average link clusters. The single link clusters also tend to be somewhat larger than the other cluster types. The theoretical basis for the differences among the hierarchies is discussed in the next

	A	B	C	D	E
B	.3				
C	.5	.4			
D	.6	.5	.3		
E	.8	.7	.5	.4	
F	.9	.8	.2	.1	.3

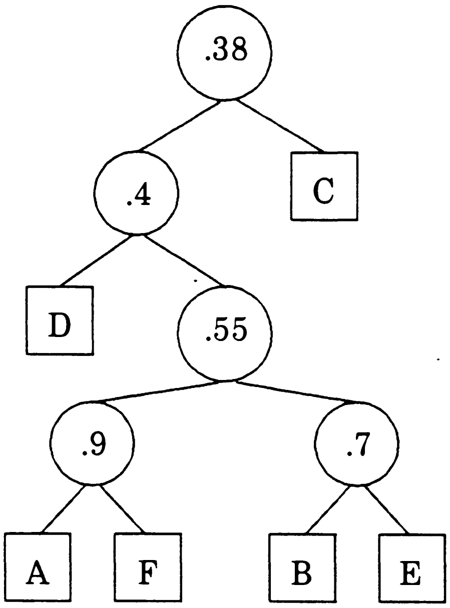
a) the similarity matrix



b) the single link hierarchy



c) a complete link hierarchy



d) a group average link hierarchy

Figure 2.2: Cluster hierarchies for a sample set of data

subsection; the differences among the hierarchies constructed for the collections used in the retrieval experiments are discussed in section 2.5.3.

Equivalent alternative definitions of the single link and complete link methods can be found in the literature [Anderberg 73]. If the similarity matrix of the documents to be clustered is represented as a weighted, undirected graph where the nodes of the graph represent the documents and the weight of an edge is the similarity of the documents connected by the edge, the single link hierarchy of the collection is also the maximum spanning tree of the graph. Alternatively, given a similarity value, v, the single link clusters at level v are precisely the connected components of the graph when all edges of weight less than v are removed. The complete link clusters at level v are the cliques (maximally connected subgraphs) of the graph when all edges of weight less than v are removed. The clusters represented in the single link and complete link hierarchies are the set of clusters obtained by varying v over all distinct values in the similarity matrix.

2.2.2. Theoretical Properties

The graph-theoretic definitions of single link and complete link clustering emphasize the differences between the clusters that are formed by these two methods. The single link method, with its easy-to-join philosophy, forms loose, "scraggly" clusters. The similarity level at which the cluster is formed implies only that each document in the cluster has that similarity level with at least one other document in the cluster; the minimum similarity between documents in the same cluster may be zero. On the other hand, each document must be in the same cluster as the document to which it is most similar (its *nearest neighbor*). Complete link clusters are difficult to join, and, therefore, are small and tight. By definition, the minimum similarity between documents in the same cluster is the similarity level at which the cluster forms, but the nearest neighbor of a document may be in a different cluster. (Mutual nearest neighbors, pairs of documents in which the nearest neighbor of each document in the pair is the other document in the pair, are always in the same clusters.) Unsurprisingly, the group average link method produces clusters which are neither as scraggly as the single link clusters nor as tight as the complete link clusters. Since the clusters are formed on the basis of average similarities, nothing can be said about the minimum or maximum similarities between documents in a cluster.

The idea of easy or difficult joining of a cluster was made more precise by Lance and Williams [Lance & Williams 67]. They describe hierarchic clustering methods in terms of three characteristics. One of the characteristics is whether a method is *space-conserving* or *space-distorting*. A method is space-conserving if, given the space defined by the similarities between documents, the properties of the space remain the same when the similarities between (non-singleton) clusters are considered. Otherwise it is space-distorting. If a method is space-distorting, it can either be *space-contracting*, where clusters appear to move closer to other clusters and thus become easier to join, or *space-dilating*, where clusters appear to recede from other clusters and become more difficult to join. The single link method is space-contracting, the complete link method is space-dilating, and the group average link method is space-conserving [Lance & Williams 67].

The other two characteristics used by Lance and Williams are whether a method is *combinatorial* or *non-combinatorial* and whether it is *compatible* or *incompatible.* A method is compatible if the similarity measure between non-singleton clusters has the exact same properties as the measure used between documents. Each of the clustering methods used in this thesis is compatible with the cosine similarity measure (the similarity measure used). A method is combinatorial if the similarity between a newly merged cluster and some other cluster can be computed from the sizes of the clusters and the similarities between the original clusters. To be more exact, let clusters p and q be the clusters that were merged to form cluster r. A method is combinatorial if for all other clusters k

$$sim(r,k) = \alpha_p sim(p,k) + \alpha_q sim(q,k) + \beta sim(p,q) + \gamma|sim(p,k)\text{-}sim(q,k)| \quad (2.2)$$

for parameters α_p, α_q, β, and γ.

Each of the clustering methods used here is combinatorial: the single link, complete link, or group average link hierarchy is that hierarchy which is pro-

Table 2.1: Values for parameters of Equation 2.2 to obtain different clustering methods

	α_p	α_q	β	γ
single link	1/2	1/2	0	$-1/2$
complete link	1/2	1/2	0	1/2
group average link	$\lvert p\rvert/\lvert r\rvert$	$\lvert q\rvert/\lvert r\rvert$	0	0

duced by computing the similarities between clusters using Equation 2.2 with the appropriate parameter values from Table 2.1. Lance and Williams showed that if $\alpha_p + \alpha_q + \beta \geq 1$, then clusters merge at monotonically decreasing similarity values. The fact that this condition holds for the group average link method (as well as the other two methods) is the reason that this particular average link method was used in this thesis.

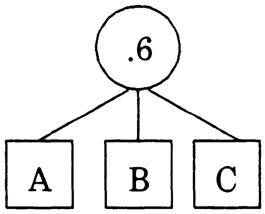
Instead of describing the properties of different clustering methods, Jardine and Sibson advanced a set of properties that all "adequate" clustering methods should possess [Jardine & Sibson 68]. Included in the set were the following:

- the clustering method should be independent of monotonic transformations of the similarity measure;
- the clustering method should be independent of the initial labeling of the items to be clustered;
- small changes in the data should produce small changes in the final hierarchy; and
- a given set of data should define exactly one hierarchy.

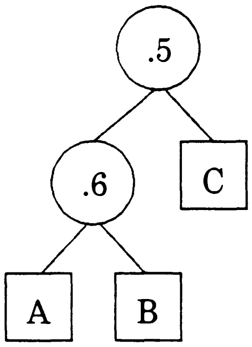
Jardine and Sibson noted that many of the well known clustering algorithms do not meet all of these criteria. Of the clustering algorithms used here, only the single link method possesses all of the properties; the complete link and group average link methods do not define an unique hierarchy for all sets of data.

The complete link and group average link methods do not necessarily define an unique hierarchy for a given set of data because ties are broken arbitrarily when there is more than one pair of clusters that have the maximum similarity. In the single link method, the merging of two clusters cannot decrease any cluster-cluster similarities, and thus ties are not a problem. However, in both of the other two methods, the similarity between a cluster, k, and a newly formed cluster, $r = merge(p,q)$, is usually smaller than the maximum of $sim(k,p)$ and $sim(k,q)$. Different similarities will be affected when ties are broken differently, and different hierarchies may result.

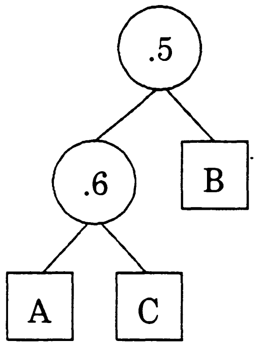
As a trivial example of the difference ties make, consider the case in which there are just three documents with inter-document similarities [A,B .6], [A,C .6], and [B,C .5]. The single link hierarchy defined by this data is



but both



and



are complete link hierarchies defined by the data. The group average link method also suffers from the difficulty that small changes in the similarities between documents may result in large differences in the resulting hierarchies.

2.2.3. Implementation of Clustering Methods

When defining the clustering methods, it was convenient to assume that the entire similarity matrix was available, implying an $O(N^2)$ time and space dependency. Indeed, any implementation of the methods which looks at inter-item similarities requires $O(N^2)$ time in the worst case. (Murtagh discusses methods for constructing the single link hierarchy by using nearest neighbors that have a time dependency of less than $O(N^2)$ [Murtagh 83]. These methods, however, assume that the vectors are of very low dimension which is not the case in information retrieval.) An $O(N^2)$ dependency is extremely expensive even for moderately large document collections. Fortunately, the document-document similarity matrix is very sparse so substantial savings over the worst case are realizable [Croft 78].

Sibson devised an implementation of the single link method that requires $O(N^2)$ time and $O(N)$ space in the worst case [Sibson 73]. Other algorithms for constructing a maximum spanning tree, and hence a single link hierarchy, using $O(N^2)$ time and $O(N)$ space in the worst case can also be found in the literature [Tarjan 83]. However, no general algorithm that takes less than $O(N^2)$ space while using $O(N^2)$ time is known for the group average link or complete link clustering methods [Murtagh 84a]. The difference between these methods is the amount of information that needs to be stored to compute the similarities between (non-singleton) clusters. As was mentioned above, for single link clustering, merging two clusters can never decrease the similarity between clusters, and thus only the cluster with the maximum similarity to a given cluster needs to be stored for that cluster. For the other two methods, however, the similarity between specific clusters is needed at unpredictable times since creating a new cluster can (and usually does) affect the similarity between other clusters and the new cluster. Therefore, the similarities either need to be recomputed when needed (which makes the total running time of the algorithm greater than $O(N^2)$), or they need to be stored (which takes $O(N^2)$ space).

The remainder of this section describes the programs used to construct the hierarehies. More detailed pseudo-code for these algorithms and the searching algorithms of section 2.3 is given in the Appendix.

2.2.3.1. The Single Link Implementation

The algorithm used to construct the single link hierarchy is an implementation of Prim's algorithm for computing maximum spanning trees [Tarjan 83]. An arbitrary document is picked and placed in the hierarchy. An array is initialized with the similarities between this document and the remaining N-1 documents. If document i is not yet in the hierarchy, the ith entry of this array will contain the maximum of the similarities between document i and each document in the hierarchy (and the document number of the document in the hierarchy with the maximum similarity). The document, d, associated with the maximum entry of the array is added to the hierarchy and the ith entry of the array is updated by taking the maximum of $sim(i,d)$ and the current value of the entry. This process is repeated until all documents are in the hierarchy.

As an example, consider the collection of Figure 2.2. If the algorithm begins by placing document A in the hierarchy, the initial state of the algorithm is as depicted in Figure 2.3a. The maximum similarity between a document in the hierarchy and a document not in the hierarchy is between documents A

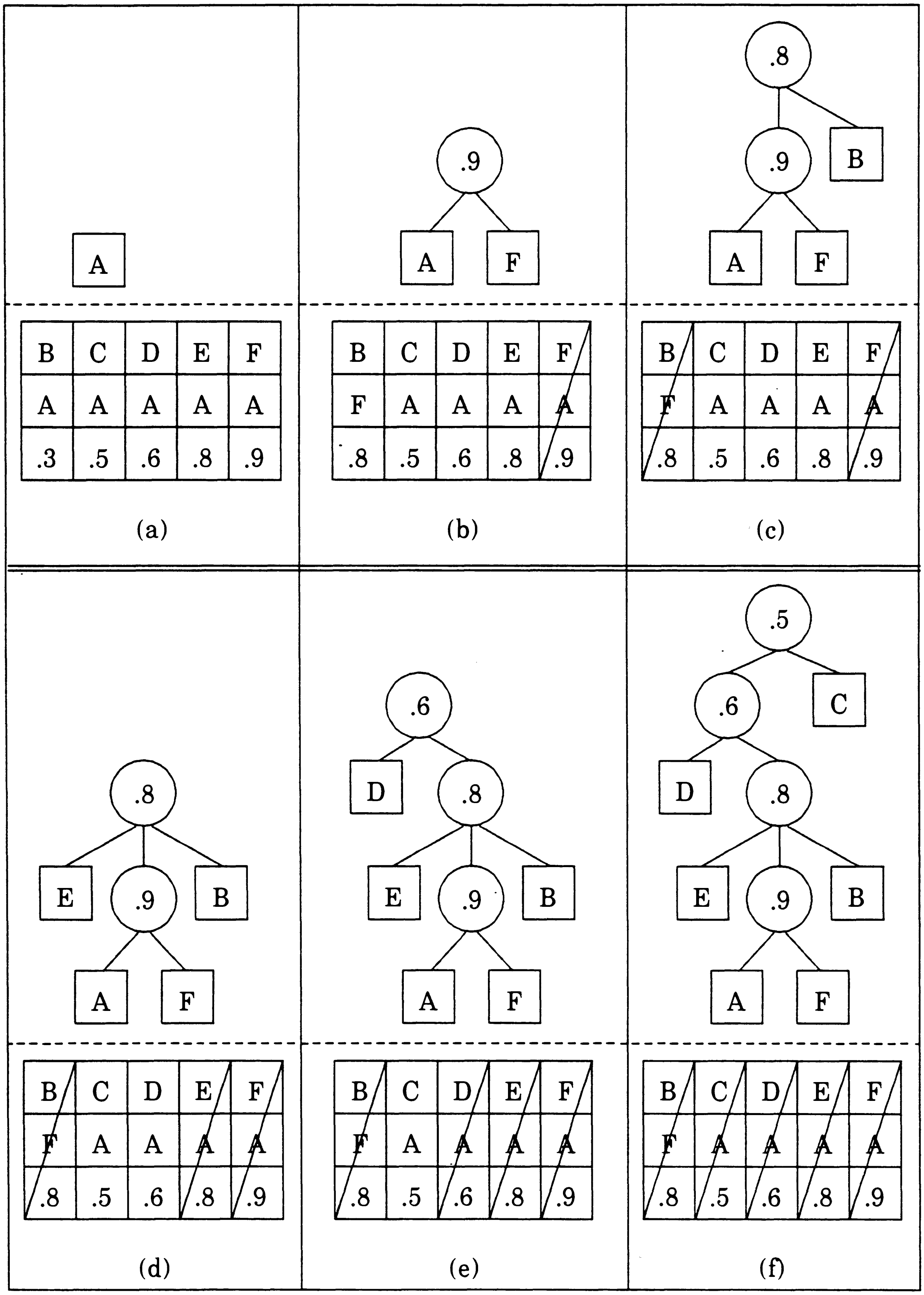


Figure 2.3: Example of constructing the single link hierarchy

and F, so document F is added to the hierarchy and the array of similarities is updated (Figure 2.3b). Since document B has a larger similarity with document F than it does with document A, its entry in the similarity array is changed; since document F is in the hierarchy, its entry in the similarity array is no longer needed. The current maximum similarity at this stage is a tie – both pairs A,E and B,F have similarity .8 – so document B is arbitrarily chosen to be added to the hierarchy (Figure 2.3c). In each subsequent iteration, the document with the maximum similarity to a document in the hierarchy is added to the hierarchy (Figure 2.3d and Figure 2.3e). The algorithm terminates when the last document is added to the hierarchy (Figure 2.3f).

The similarity used in this process was the cosine of the angle between the two document vectors, rounded to three decimal places. The cosine of the angle between two vectors, v and **w**, is

$$cos(\mathbf{v},\mathbf{w}) = \frac{\sum \mathrm{v}_i \mathrm{w}_i}{\left(\sum(\mathrm{v}_i)^2 \times \sum(\mathrm{w}_i)^2\right)^{1/2}}.$$

Using appropriately normalized weights within the vectors, the cosine similarity can be computed using the inner product function: $\sum \mathrm{v}_i \mathrm{w}_i$. The inner product has the advantage that the similarities between a given document and all other documents can be computed efficiently without computing similarities of zero [Smeaton & van Rijsbergen 81].

2.2.3.2. The Group Average Link Implementation

Using the cosine similarity measure (or any other measure that can be computed using the inner product function given appropriately weighted vectors), the group average link clustering method can be implemented using $O(N^2)$ time and $O(N)$ space in the worst case. For the inner product similarity function, the similarity between a centroid of a cluster and a document is equal to the mean similarity between the document and all the documents in the cluster. Here the centroid of a cluster is the mean of all the vectors in the cluster. Thus the centroids of the cluster can be used to compute the similarities between the clusters while requiring only $O(N)$ space. This method was employed in an adaptation of Anderberg's stored data approach algorithm [Anderberg 73, p. 146] to construct the group average link hierarchies used in this work.

The algorithm keeps a list of *active* clusters: those clusters which have not yet merged with another cluster. For example, if $r = merge(p,q)$, then before the merger p and q are active clusters (and r does not exist). After the merger, p and q are not active and r is active. Information is kept on all active clusters; this information includes the centroid of the cluster, the size of the cluster, the active cluster to which it is most similar, and the similarity it has with that cluster. At the beginning of the algorithm, each document is the centroid of its singleton cluster and all (singleton) clusters are active. The maximum similarity between active clusters is initialized by computing all non-zero document-document similarities.

The clusters that have the maximum similarity with one another are merged. That is, the cluster ids of the clusters being merged are replaced by the id of the new cluster in the active cluster list, a new centroid is computed from the original two centroids, the new size is set to the sum of the old sizes, and the new cluster is represented in the hierarchy tree. The list of active clusters is then traversed. For each active cluster, a, that has one of the two clusters that were merged as its most similar cluster (and for the cluster just created), a new most similar cluster is found by computing the similarity of the centroid of a with all other active clusters' centroids. The clusters with the maximum similarity are also determined in this traversal. The process repeats until there is only one active cluster remaining.

Figure 2.4 contains an example of constructing a group average link hierarchy using the algorithm described above. Once again, the collection of Figure 2.2 is used in the example. In the example the singleton clusters are assigned the identifiers 1-6, with the cluster consisting of document A as cluster 1, the cluster consisting of document B as cluster 2, *etc.* When two clusters are merged, the smaller of the two identifiers is used to name the newly created cluster.

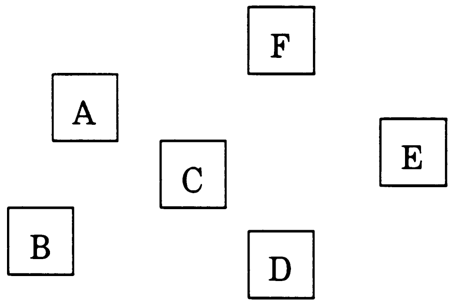
The initial state is depicted in Figure 2.4a; each cluster is active and the hierarchy is a forest of six nodes. Clusters 1 and 6 are the clusters with maximum similarity, so those clusters are merged as shown in Figure 2.4b. Since cluster 5 is most similar to cluster 3 and neither cluster 5 nor cluster 3 is involved in the merger, the information about cluster 3 is not updated in this iteration. All of the other active clusters have either cluster 1 or cluster 6 as its most similar cluster, however, so a new most similar cluster is found for those clusters. The most similar cluster to a given cluster, say cluster 2, is found by taking the maximum of the similarities between the centroid of cluster 2 and the centroids of all other active clusters. The centroids are not shown

ACTIVE: 1, 2, 3, 4, 5, 6

CLUSTER	1	2	3	4	5	6
MOST SIM	6	6	5	1	1	1
SIM	.9	.8	.5	.6	.8	.9
SIZE	1	1	1	1	1	1

merge clusters 1 and 6

(a)

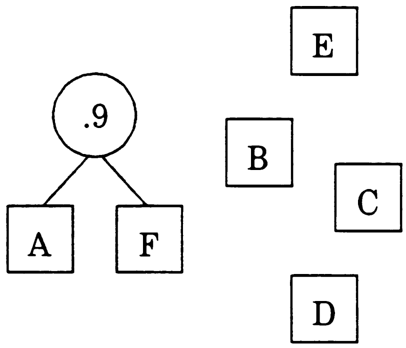


ACTIVE: 1, 2, 3, 4, 5

CLUSTER	1	2	3	4	5
MOST SIM	2	5	5	2	2
SIM	.55	.7	.5	.5	.7
SIZE	2	1	1	1	1

merge clusters 2 and 5

(b)



ACTIVE: 1, 2, 3, 4

CLUSTER	1	2	3	4
MOST SIM	2	1	2	2
SIM	.55	.55	.45	.45
SIZE	2	2	1	1

merge clusters 1 and 2

(c)

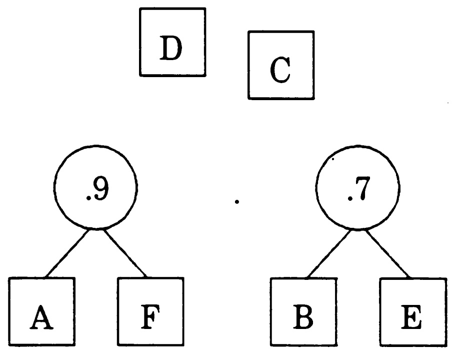


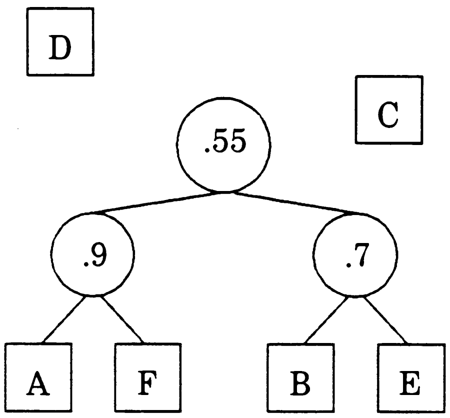
Figure 2.4: Example of constructing a group average link hierachy

ACTIVE: 1, 3, 4

CLUSTER	1	3	4
MOST SIM	4	1	1
SIM	.4	.4	.4
SIZE	4	1	1

merge clusters 1 and 4

(d)

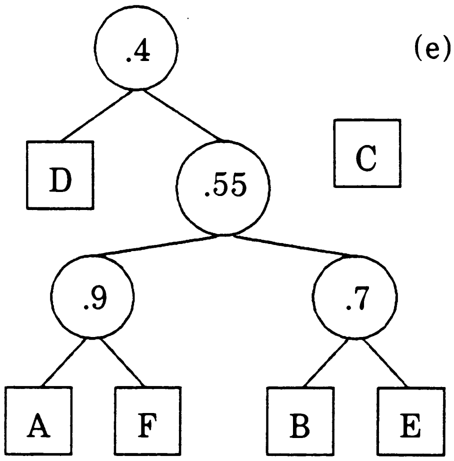


ACTIVE: 1, 3

CLUSTER	1	3
MOST SIM	3	1
SIM	.38	.38
SIZE	5	1

merge clusters 1 and 3

(e)



ACTIVE: 1

(f)

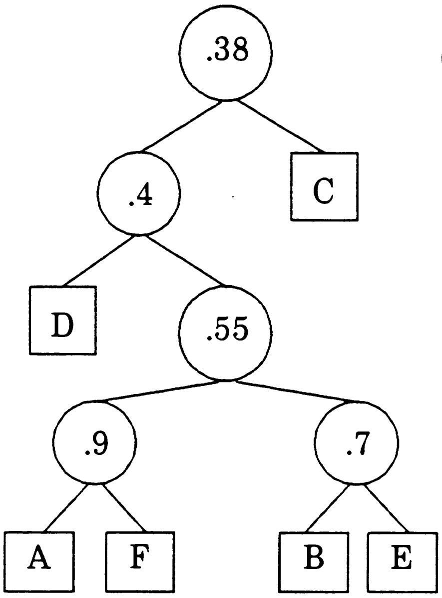


Figure 2.4 (continued)

in the example, although the new most similar cluster and corresponding similarity are shown.

After the second iteration (Figure 2.4c), clusters 2 and 5 have been merged and each of the entries in the most similar cluster array have been updated. The two most similar clusters merge in the subsequent iterations until only cluster 1 is active (Figure 2.4f).

The above algorithm is idealized somewhat to make it easier to understand. An obvious refinement is to only compute the similarities with clusters that have larger (or smaller) ids. If this optimization is used, the cluster listed as the most similar cluster may in fact not be the most similar cluster, but in this case the true nearest neighbor will have the relationship recorded accurately.

2.2.3.3. The Complete Link Implementation

The algorithm used to construct the complete link hierarchy is due to Chris Buckley. In the worst case the algorithm requires $O(N^2)$ space and more than $O(N^2)$ time, but the worst case would not be expected to arise in an information retrieval application due to the number of inter-document similarities equal to zero. The algorithm is based on the fact that two complete link clusters merge at the level of the minimum similarity between any pair of documents such that one document of the pair is in one cluster and the other document is in the other cluster. If the similarities between all pairs of documents are processed in descending order, then two clusters of size s_1 and s_2 can be merged as soon as the $s_1 \times s_2$th similarity of documents in the respective clusters is reached. This requires two things: a sorted list of document-document similarities and a method to keep track of the number of similarities seen between any two active clusters.

The complete link algorithm alternates between two phases, generating and sorting similarities, and using the sorted similarities to construct the hierarchy. In *each* sorting phase all non-zero similarities are computed, but only the next k similarities are put out. The value of k depends on two factors: a small k implies short sorting times and a smaller amount of space needed by the sorting routine. However, it also implies all the similarities will have to be recomputed more times since the number of times the similarities will be recomputed is N/k (thus the worst case running time is $O(N^3)$). In the second phase the current number of similarities between active clusters is kept in a set of linked lists. Associated with each active cluster is the size of the cluster and a linked list of other active cluster ids with the number of similarities that have been seen between the current cluster and the clusters on the list. (In the worst case, $O(N^2)$ linked list entries will be required.) Associated with each document is the id of the active cluster that contains the document. Upon reading an input triple – two document ids and their similarity – the program increments the count of the number of similarities seen between the appropriate clusters. If this is not the last similarity for those clusters, the next input triple is read. If it is the last similarity, the clusters are merged.

Merging clusters entails keeping the data structures up to date in addition to actually representing the new cluster in the hierarchy. The documents in the two merged clusters have their active cluster id updated. The list of clusters associated with the new cluster is the union of the lists of the merged clusters. This phase ends when the last similarity is read; the entire algorithm ends when the last non-zero similarity is processed.

An example construction of a complete link hierarchy for the collection of Figure 2.2 is shown in Figure 2.5. In the example, three is used for the value of k in phase one of the algorithm.

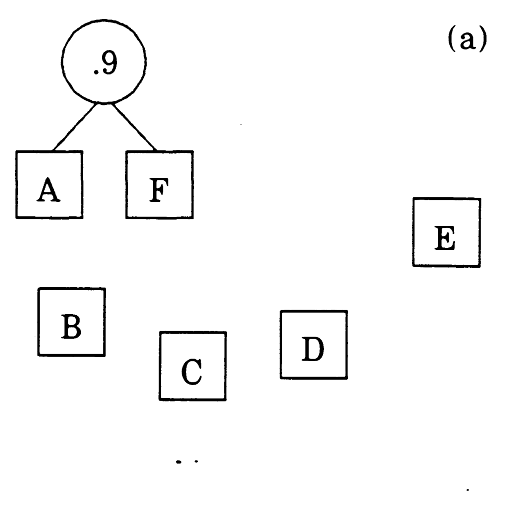
Figure 2.5a depicts the state of phase two of the algorithm after the three greatest similarities have been read. (The links field is of the form [cluster id, number of similarities seen].) Since clusters 1 and 6 each contain one document, the first similarity causes those two clusters to merge. The next two similarities each link one other cluster (clusters 2 and 5, respectively) to cluster 1. Cluster 1 now contains two documents, however, so $2 \times 1 = 2$ similarities must be seen before a singleton cluster may merge with cluster 1.

In Figure 2.5b, the next two similarities have been put out by phase one of the algorithm. In order to keep track of the maximum similarity that should be considered in the next iteration, phase one writes out all equal similarities in the same iteration. Since no more than k ($= 3$) similarities may be written out in any iteration, only two similarities are processed in this iteration.

The first of the two similarities links singleton clusters 2 and 5, causing the clusters to merge. In the process of merging the two clusters, the list of links for all other clusters linked to the merged clusters (in this case cluster 1) must be updated. Since cluster 2 had one link to cluster 1 and cluster 5 had one link to cluster 1, the newly formed cluster 2 has two links to cluster 1. The second similarity links clusters 1 and 3.The next three similarities, shown in Figure 2.5c, do not cause any mergers. The first similarity of the iteration depicted in

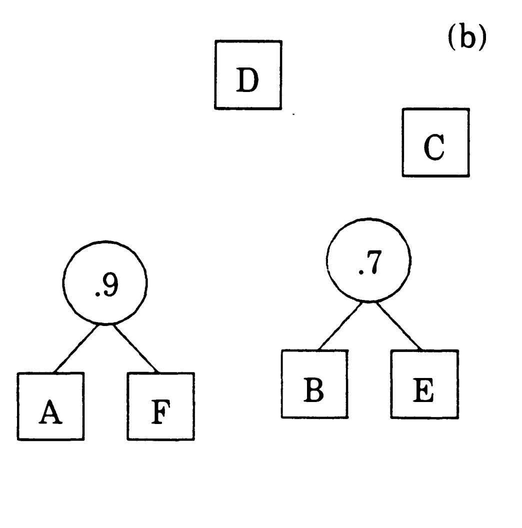
Phase 1: [A,F .9] [A,E .8] [B,F .8]

ID	Contents	Links
1	A,F	[2,1] [5,1]
2	B	[1,1]
3	C	--
4	D	--
5	E	[1,1]



Phase 1: [B,E .7] [A,D .6]

ID	Contents	Links
1	A,F	[2,2] [4,1]
2	B,E	[1,2]
3	C	--
4	D	[1,1]



Phase 1: [A,C .5] [B,D .5] [C,E .5]

ID	Contents	Links
1	A,F	[2,2] [4,1] [3,1]
2	B,E	[1,2] [4,1] [3,1]
3	C	[1,1] [2,1]
4	D	[1,1] [2,1]

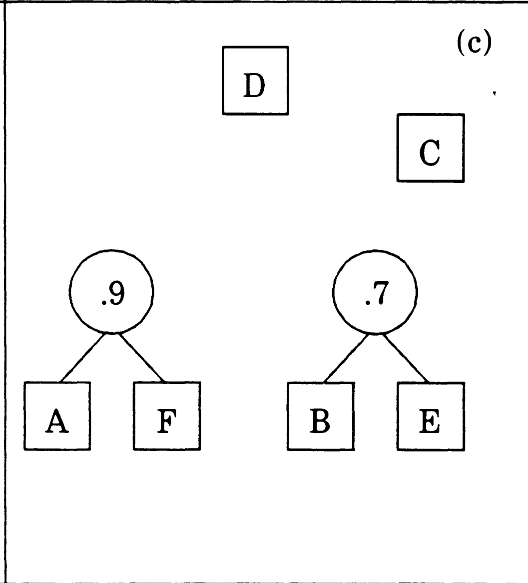
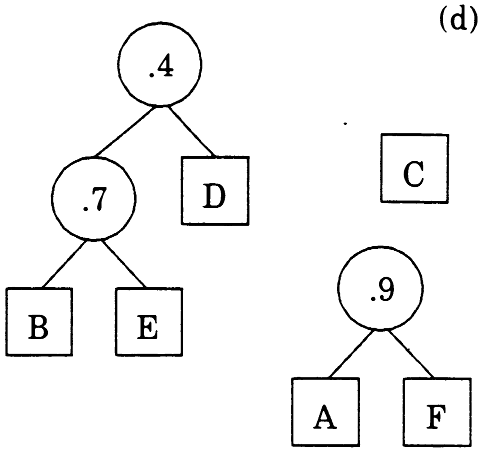


Figure 2.5: Example of constructing a complete link hierachy

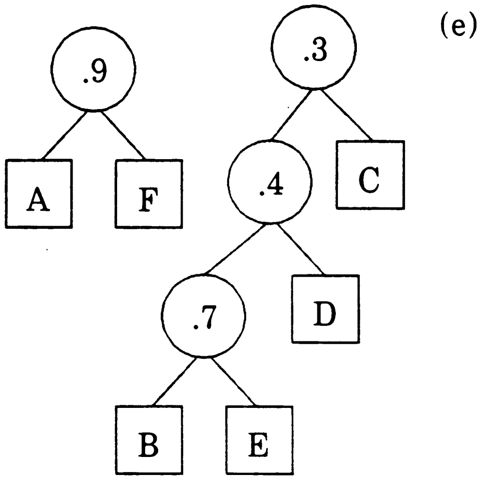
Phase 1: [D,E .4] [B,C .4]

ID	Contents	Links
1	A,F	[2,3] [3,1]
2	B,D,E	[1,3] [3,2]
3	C	[1,1] [2,2]



Phase 1: [A,B .3] [C,D .3] [E,F .3]

ID	Contents	Links
1	A,F	[2,3] [3,1]
2	B,C,D,E	[1,3] [3,2]



Phase 1: [D,F .1]

ID	Contents
1	A,B,C,D,E,F

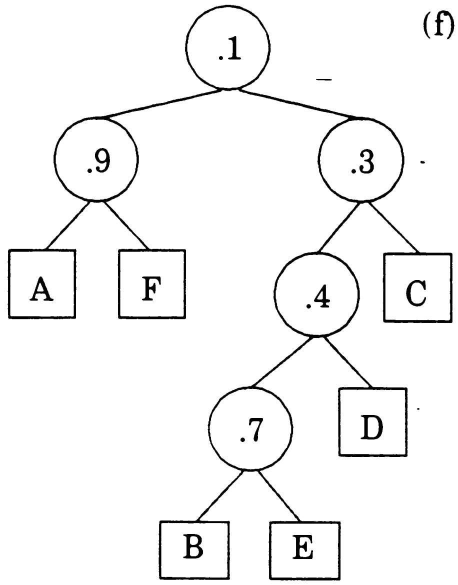


Figure 2.5 (continued)

Figure 2.5d, however, is the last similarity to be seen between clusters 2 and 4, causing those clusters to be merged. Clusters 2 and 3 are merged in the next iteration, Figure 2.5e, and the final two clusters merge in the last iteration. The algorithm terminates after the last similarity ([D,F .1]) has been processed.

2.3. The Searches

The creation of a cluster hierarchy is a preliminary step to performing cluster searches. This section defines the four cluster searches that will be investigated on each of the hierarchy types. (More detailed descriptions are given in the Appendix.) The four searches result from varying two variables: retrieving entire clusters or individual documents from within clusters and searching top-down or bottom-up. The inverted index search, the search that is used as a benchmark for the performance of the cluster searches, is also described.

2.3.1. The P-norm Inverted Index Search

Conceptually, the inverted index search is identical to a sequential scan of the document collection since a total ranking of the collection is produced. However, the inverted index search is much more efficient than a sequential scan since only those similarities that are guaranteed to be non-zero are computed. The search is performed as follows.

1) The inverted lists for each query term are brought into main memory. (Note that each inverted list entry requires only eight bytes – four bytes for the document id and four bytes for the weight. Since only the query terms' inverted lists are brought into main memory, total memory usage remains modest even when high frequency terms are included in the

query.) The lists are assumed to be sorted by increasing document id (*did*) and to contain the weight of the term in the document. Associated with each list is a pointer which points to the current *did* for that list. This pointer is initialized to point to the first *did* in the list.

2) The minimum of the *did*s being pointed to by the list pointers is the (global) current *did*, and the corresponding document is the current document. The procedure which computes the similarity between a single document and a query is called with the weight of each of the query terms in the current document supplied as parameters. The weights are obtained by checking the list pointer of each inverted list. If the list pointer points to the current *did*, the weight in that inverted list entry is used and the list pointer is advanced. If the list pointer points to another *did*, the weight of the query term corresponding to the list is zero in the current document.

3) Step 2 is repeated until all list pointers have reached the end of their lists.

Figure 2.6 contains an example search for the query Q = A **or** (E **and** F). Figure 2.6a depicts the initial configuration. The inverted list (in the form [did, weight]) are shown for each of the three query terms. The pointer of each list points to the first *did* in the list.

The minimum *did* being pointed to is 2. Thus document 2 becomes the current document. Since the only query term to appear in document 2 is term A, the evaluation procedure is called with (A=.7, E=0, F=0), and term A's list pointer is advanced.

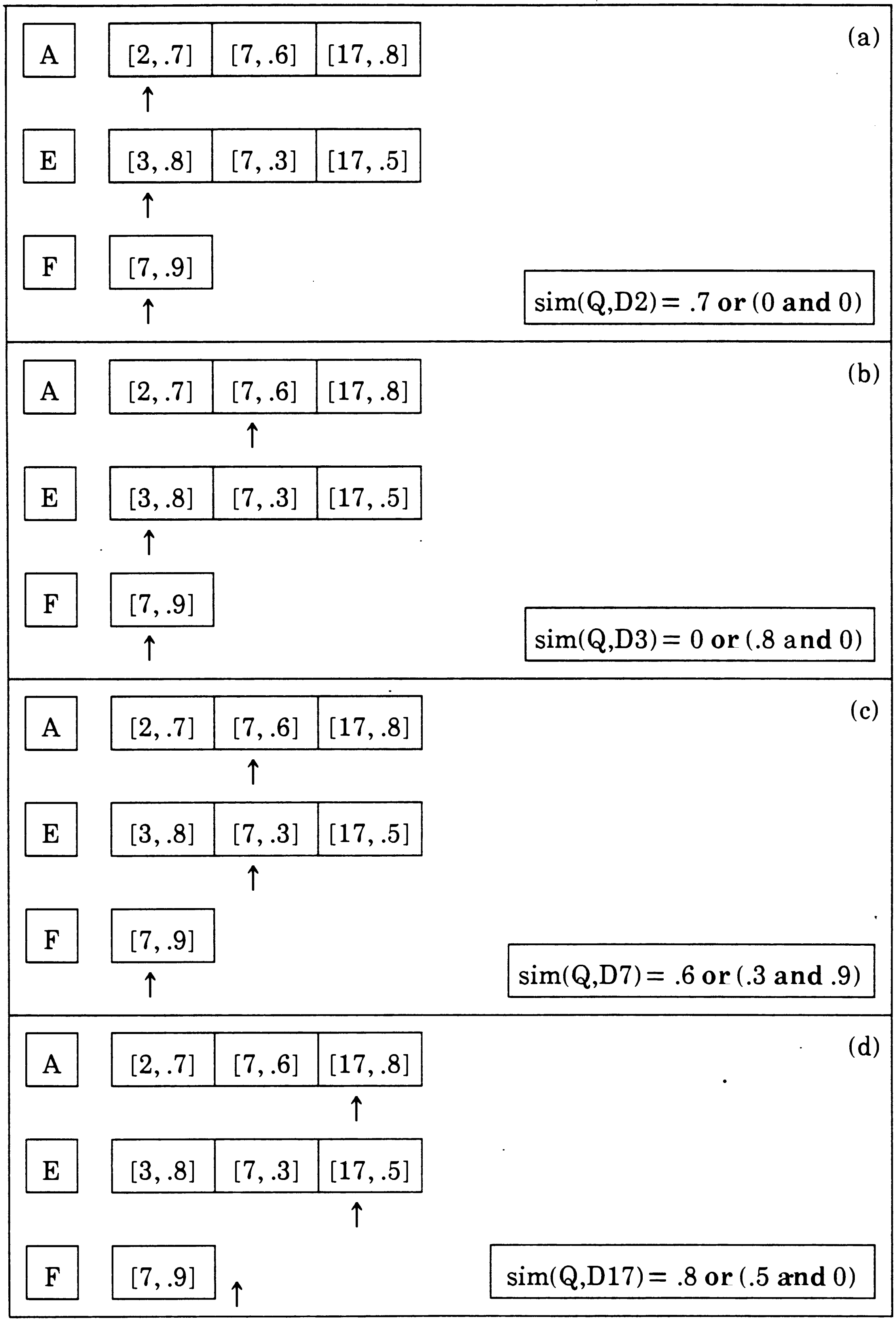


Figure 2.6: An inverted index search for the query A **or** (E **and** F)

Figure 2.6b depicts the state after the similarity between the query and document 2 has been computed. The new current document is document 3. Once again only one query term, term E, appears in the current document. The evaluation procedure is called with (A=0, E=.8, F=0), and term E's list pointer is advanced.

After term E's list pointer has been advanced (Figure 2.6c), all three list pointers point to the same *did*, signifying that all three query terms appear in document 7. The evaluation routine is called with (A=.6, E=.3, F=.9), and all three list pointers advance. Term F's list pointer advances past the end of the list, implying that term F appears in no more documents.

The evaluation procedure is called once more (for document 17) until all lists have been exhausted. Note that the similarity evaluation procedure is called once for each *did* in the union of the query terms' inverted lists.

2.3.2. Cluster Searches

Each of the cluster searches requires centroids as representatives of the clusters. The purpose of these centroids, facilitating searching, is quite different from the use of centroids in constructing the group average link hierarchy, and in consequence the centroids are defined differently. The centroids used for searching are based on the rank weight centroids of Murray [Murray 72]. Specifically, the following steps are performed to create the centroid of a cluster.

1) For each term appearing in a document in the cluster, the sum of the term frequency weights of that term in all the documents of the cluster

is computed. The terms are then ranked from largest to smallest by their total frequency.

2) Those x terms that have the greatest frequency are selected to be in the centroid. The weight of a term in the centroid is then defined as its rank (from the bottom) in the sorted list – equal frequencies are assigned the same rank. These are the weights used in Murray's P_3* centroids.

3) These rank weights are then used as the term frequency component of the weighting function described in Equation 2.1. (The inverse document frequency component is still based on the number of documents, not centroids, in which the term appears.) The weights of the terms are further normalized so that the sum of the squares of the weights equals one to produce the final centroid.

Each of the cluster searches also requires an argument, called *NumWanted*, that specifies the approximate number of documents that should be retrieved. The number of clusters that are explored depends on this parameter. In general, retrieving more documents will produce better recall but also be more costly than retrieving fewer documents.

The top-down search that retrieves clusters in their entirety, **td-entire**, begins by placing the root of the hierarchy into an empty heap. While the heap is not empty and fewer than *NumWanted* documents have been retrieved, the top of the heap (corresponding to the cluster with the current largest similarity with the query) is popped. If the cluster represented by the popped node is small enough – if the size of the cluster is smaller than the difference between the number of documents already retrieved and *NumWanted* plus a small constant – all the documents in that cluster are retrieved. Otherwise, the non-leaf children of the node that have non-zero similarities with the query are added to the heap and the process repeats. The search terminates when enough documents have been retrieved or the heap is empty.

The top-down search that retrieves individual documents from within clusters, **td-indiv**, is similar. Once again, the search begins with the root being added to an empty heap, and the top of the heap is popped until sufficient documents have been retrieved or the heap is empty. If the node popped from the heap is a leaf, the document represented by that node is retrieved. Otherwise, all the children of the node that have non-zero similarities with the query are added to the heap.

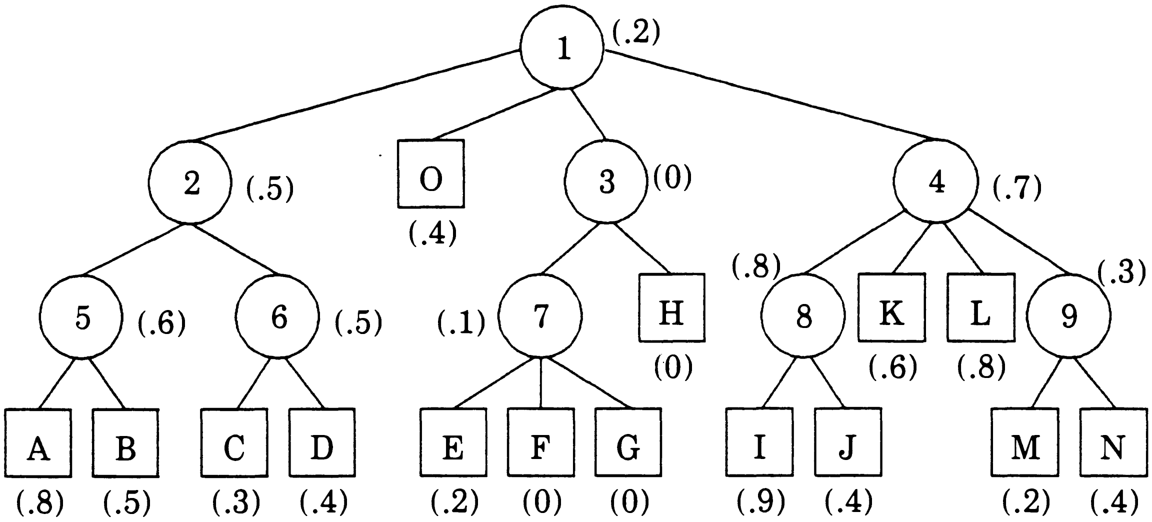
Since the path down the tree is controlled by the nodes that are popped from the heap, the search is neither a true depth-first nor a true breadth-first search. In practice, it is closer to a depth-first search (with some backtracking) than to a breadth-first search.

The two bottom-up searches use an inverted index of the low-level centroids as an entry into the hierarchy. (Recall that the low-level clusters are the set of clusters such that each cluster is the smallest cluster to contain some document.) The searches begin by performing a traditional inverted index search on the low-level centroids to locate the x most similar centroids. In the bottom-up search that retrieves entire clusters, **bu-entire**, the documents within the top-ranked clusters are then retrieved until approximately *NumWanted* have been retrieved. In the bottom-up search that retrieves individual documents from within clusters, **bu-indiv**, all the documents that are contained within the top ranked clusters are ranked by decreasing similarity to the query and the top *NumWanted* are returned to the user. In both searches, if the size of a highly ranked cluster is too large (greater than *NumWanted* plus a pre-defined constant), only the immediate sub-clusters that are small enough are used in the retrieval.

All of the searches rank the documents that are returned to the user. In the searches that retrieve individual documents, the similarity between the query and the documents is already computed and is used to rank the documents. For the entire cluster searches, individual document similarities are never computed, so they cannot be used to rank the documents. Instead, the documents within the same cluster are assigned the same similarity; the similarity assigned is inversely proportional to the rank of the cluster which contains them.

A sample search for each of the different searching algorithms is given in Figure 2.7. 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 (hypothetical) query. The value of *NumWanted* and the number of centroids returned in the bottom-up searches are both three. A cluster is considered to be too big if it has more than four documents in it.

The samples illustrate some of the properties of the searches. The searches that retrieve individual documents return exactly *NumWanted* documents, although the number of similarities computed is usually much greater than



bu-entire	**bu-indiv**	**td-entire**	**td-indiv**
Centroids returned: 8,4,5 Retrieve I,J from cluster 8 Cluster 4 too big; retrieve small enough children until *NumWanted* docs retrieved Retrieve K from cluster 4. ——— Docs retrieved: I sim=1.0 J sim=1.0 K sim=.9	Centroids returned: 8,4,5 Rank documents in those clusters: I, .9 B, .5 L, .8 J, .4 A, .8 N, .4 K, .6 M, .2 Return top *NumWanted*. ——— Docs retrieved: I sim=.9 L sim=.8 A sim=.8	pop [1, .2] too big so add [2, .5], [4, .7] pop [4, .7] too big so add [8, .8], [9, .3] pop [8, .8] retrieve I, J pop [2, .5] too big so add [5, .6], [6, .5] pop [5, .6] retrieve A, B ——— I sim=1.0 J sim=1.0 A sim=.9 B sim=.9	pop [1,.2] add [2, .5], [4, .7], [O, .4] pop [4, .7] add [8, .8], [9, .3], [L, .8] [K, .6] pop [L, .8] pop [8, .8] add [I, .9], [J, .4] pop [I, .9] pop [K, .6] ——— I sim=.9 L sim=.8 K sim=.6

(*NumWanted*=3; cluster size constant=1; number centroids retrieved=3)

Figure 2.7: Sample cluster searches

that. The entire searches may return slightly more than *NumWanted* documents, but they compute far fewer similarities than the individual documents searches since no query-document similarities are computed. The **td-entire** search cannot retrieve any document whose low-level cluster is too large (such as documents K, L, and O in the Figure). In effect these documents are no longer part of the collection.

None of the searches retrieves a single "best" cluster as was done in earlier work for two reasons: without restrictions on the minimum and maximum size of the cluster to be retrieved, the number of documents returned to the user is uncontrollable and no ranking of the documents that are returned can be done. These are two of the major disadvantages of the pure Boolean systems. Little can be gained by repeating these flaws in a cluster environment.

2.4. Evaluation of Clustered Retrieval

The standard way of evaluating the effectiveness of different retrieval methods is to perform experiments using test collections. These collections consist of a set of documents and a given set of queries for those documents. Associated with each query is a list of the documents that are relevant to that query. The same queries can then be submitted to each of the competing retrieval systems to be scored against the same set of documents. The sets of documents returned by the systems (alternatively, the rankings of the collection produced by the systems) can then be directly compared.

Given a ranking of the document collection, the most common way of measuring the effectiveness of the system that produced the ranking is through a recall and precision graph [Salton 71a]. In these graphs the precision is plotted at standard recall levels. As recall and precision are inversely related in practice, the recall and precision graph is a method of seeing how the precision of the search varies with the recall.

Clustered retrieval is not amenable to the creation of recall and precision graphs since a full ranking of the document collection is not produced. Instead the evaluation measures are computed at document levels (*i.e.* after a given number of documents have been retrieved). For a specific number of retrieved documents, the relative effectiveness of a set of runs is identical when measured by either the recall or the precision of the retrieved sets: both the recall and the precision of a set of documents increase as the number of relevant documents included in the set increases. Therefore, only one of recall and precision needs to be computed in order to make the comparisons.

Each of the searches described in section 2.3 produces a ranking of some portion of the document collection (for the inverted index search all the non-zero similarities are computed so in principle the entire collection can be ranked by decreasing similarity to the query; for the clustered searches *NumWanted* documents are ranked). Define the retrieved set to be the first x documents in the ranking. Then the effectiveness measures used in this thesis are the mean precision of the retrieved set over all the queries, the total number of relevant documents in the retrieved sets of all the queries, and the total number of queries that contain no relevant documents in their retrieved sets.

2.5. The Test Collections

Just as each part of an information retrieval system may affect the performance of all other parts, the characteristics of a particular document collection may also affect both retrieval effectiveness and efficiency. For this reason, experiments need to be run on several test collections to demonstrate that the results of the experiments are generally valid. Furthermore, examination of the characteristics of test collections frequently provides insight into the retrieval process.

2.5.1. Characteristics

Each of the experiments in this thesis was run on four document collections. The general characteristics of each of these collections can be found in Table 2.2. The CACM collection, consisting of 3204 documents and 50 queries in computer science, and the CISI collection, consisting of 1460 documents and 35 queries in information science, were created by Fox [Fox 83b]. (The document vectors of these collections contain more information, such as author names and citation information, than the standard content indicators. To remain compatible with the other collections, only the standard content indicators were used in the experiments. This required removing two of the queries from the CACM collection since those queries contained only non-content indicators.) The MED collection consists of 1033 documents and 30 queries in biomedicine and was obtained with help from the National Library of Medicine. The INSPEC collection consists of 12684 documents and 77 queries in electrical engineering and is used with permission of The Institution of Electrical Engineers, Hertfordshire, England.

The collections have widely varying properties. CACM has few relevant documents per query while CISI has extraordinarily many. Thus it is likely that for a given number of retrieved documents, CISI searches will have high

Table 2.2: 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 terms per query	4.5	7.2	13.2	39.7
mean docs per query term	150.2	225.9	950.0	28.1
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

precision but low recall, and *vice versa* for CACM. CACM also has few terms per query while MED has many. The mean number of terms per query and the mean number of documents in which query terms appear are important factors in the efficiency of inverted index searches. This point will be developed in detail in Chapter 4.

2.5.2. Characterization by the Cluster Hypothesis

For test collections, it is possible to examine the extent to which the cluster hypothesis characterizes the collection. Jardine and van Rijsbergen introduced the *cluster hypothesis test* as one method of determining this fact [Jardine & van Rijsbergen 71]. The test involves plotting two frequency distributions and observing the difference between them. The distributions to be plotted are the frequency distribution of the similarities between all pairs of documents such that both of the documents are relevant to the same query, and the frequency distribution of the similarities between all pairs of documents such that one document is relevant to some query and the other document is not relevant to that query. A separation between the two distributions implies the cluster hypothesis may be true for the collection. The plots for the four test collections can be found in Figure 2.8. The separation between the frequency distributions for the MED, CACM, and INSPEC collections is substantial, while the separation for the CISI collection is quite small.

This test has been useful in explaining the widely varying effect of changes in retrieval strategy for different collections [van Rijsbergen & Sparck Jones 73]. However, its appropriateness for testing the cluster hypothesis is open to question. There are always many more relevant-nonrelevant than relevant-relevant pairs, causing the relative frequency of very similar relevant-nonrelevant pairs to be much smaller than the relative frequency of very similar relevant-relevant pairs, even if the absolute number of nonrelevant pairs is the same. Whether or not the cluster hypothesis characterizes a collection does not depend on the relative frequency, but rather on the absolute number of nonrelevant documents that are very similar to the relevant documents. Since the cluster hypothesis test does not give information at this level of detail, another test, the *nearest neighbor test*, was performed on the document collections.

If the cluster hypothesis characterizes a collection, many of the nearest neighbors of a relevant document will also be relevant. The nearest neighbor test checks if this condition holds by computing the n nearest neighbors of a

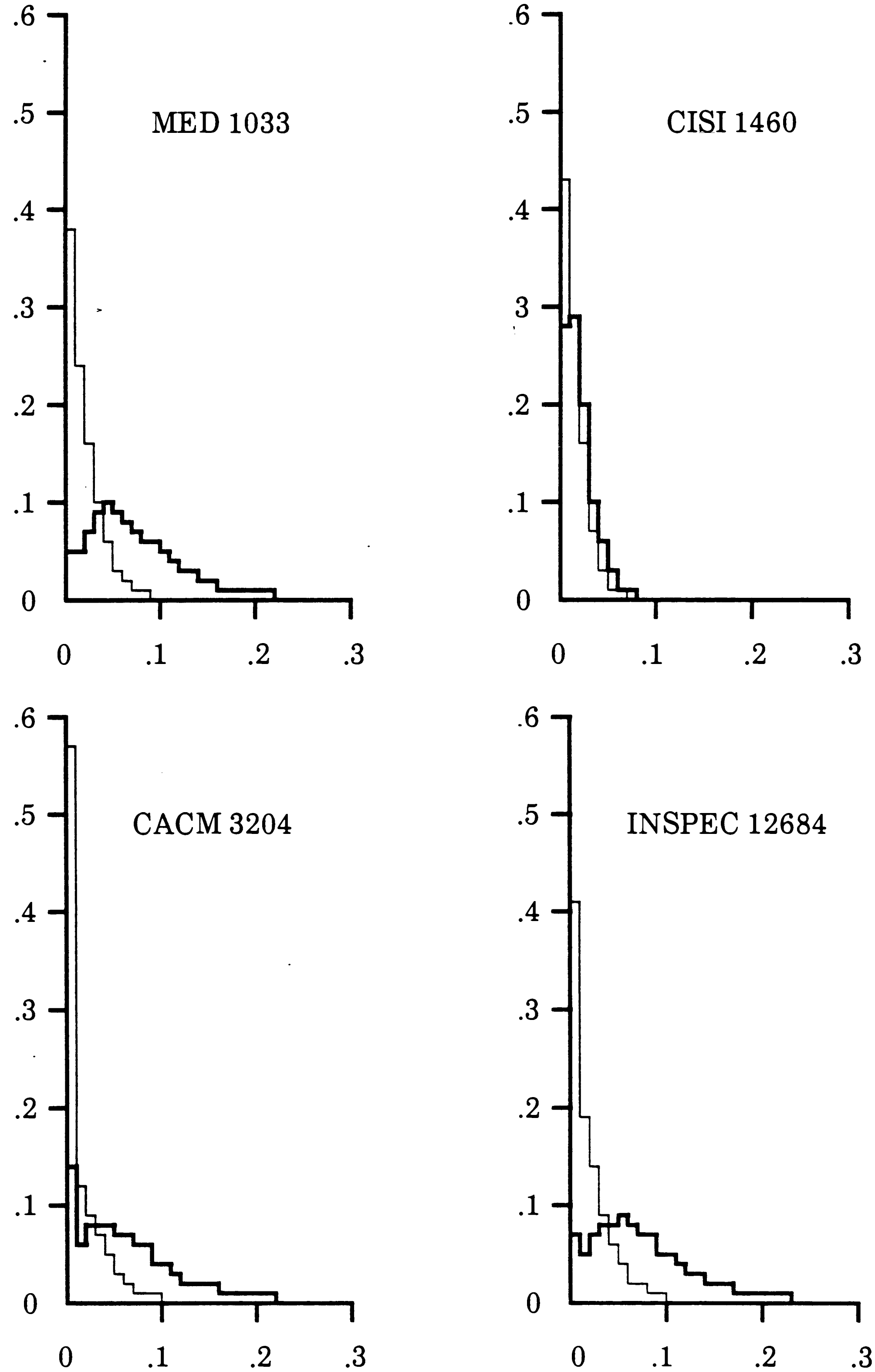


Figure 2.8: Cluster hypothesis test results. The thin line is the frequency distribution of the similarities between relevant-nonrelevant pairs; the heavy line of the relevant-relevant pairs

Table 2.3: Percentage of relevant documents with given number of relevant nearest neighbors

# rels in nearest neighbor set	CACM	CISI	INSPEC	MED
0	28	38	46	8
1	29	30	24	11
2	20	20	14	17
3	15	8	8	23
4	5	3	5	24
5	3	1	4	17

relevant document and recording the number of these documents that are also relevant. This process is repeated for each relevant document of each query that has more than one relevant document.

The results of the nearest neighbor test for each of the four collections can be found in Table 2.3. For each of the collections the value of n was (arbitrarily) set to five. The Table gives the percentage of relevant documents that have 0, ..., 5 relevant nearest neighbors. For example, the first entry in the Table states that for 28% of the relevant documents of the CACM collection, the relevant document has no other relevant document in the set of the five documents most similar to it. Similarly, for 3% of the relevant documents of the CACM collection, the five documents most similar to the relevant document are also relevant.

Two conclusions may be drawn from the results of the nearest neighbor test: the cluster hypothesis characterizes the MED collection to a much larger extent than it does the other collections (41% of the relevant documents have at least four relevant documents among the five nearest neighbors for MED as opposed to at most 9% for the other collections), and, for all collections except MED, many relevant documents are more similar to nonrelevant documents than they are to other relevant documents. The latter conclusion was also reached by Ide in her work on relevance feedback [Ide 69].

The cluster hypothesis test and the nearest neighbor test give rather different pictures of the document collections. The INSPEC collection has a reasonably good separation of the distributions in the cluster hypothesis test. In contrast, the nearest neighbor statistics show that for nearly half the relevant documents, there are no other relevant documents among the five nearest neighbors, and that for 70% of the relevant documents there is at most one relevant document among the five nearest neighbors. The CACM collection, which also has a good separation in the cluster hypothesis test, has 57% of the relevant documents with at most one other relevant document among the five nearest neighbors. The distribution of the MED collection in Figure 2.8 is similar to the distributions of the CACM and INSPEC collections, but MED has only 19% of the relevant documents with at most one other relevant document among the five nearest neighbors. The two tests agree on the CISI collection: that collection has a small separation of the frequency distributions and correspondingly poor percentages of relevant documents with relevant nearest neighbors.

The percentage of relevant documents that have other relevant documents as nearest neighbors gives a more accurate description of how well the cluster hypothesis characterizes a collection than the separation of the frequency distributions of the similarities between relevant-relevant and relevant-nonrelevant documents does. The test results suggest that the cluster hypothesis holds for the MED collection and does not hold for the CISI collection. It characterizes the other two collections to a limited extent.

2.5.3. Hierarchy Statistics

The characterization of the document collections by the cluster hypothesis depends only on the similarity measure used and the queries and relevance assessments, not on a particular clustering method. In this section the characteristics of the hierarchies produced by the three different clustering methods will be examined. These statistics are useful not only for displaying the differences among the collections but also the differences among the hierarchy types.

Tables 2.4-2.6 contain the statistics for the single link, group average link, and complete link hierarchies, respectively. The *depth* of a hierarchy is the length of the longest path from the root of the tree that represents the hierarchy to a leaf node. *Interior nodes* are non-leaf nodes. *Twigs* are interior nodes that have a leaf as a child. The low-level clusters are precisely those clusters represented by twigs, so the number of twigs is also the number of low-level centroids that need to be inverted for the bottom-up searches. (The percentage of twigs given is the percentage of all interior nodes that are twigs.) The "max children" and "mean children" entries refer to the maximum

Table 2.4: Single link hierarchy statistics

	CACM	CISI	INSPEC	MED
depth	389	143	306	141
max children	17	20	79	11
mean children	2.92	3.00	3.67	2.50
# interior nodes	1669	730	4742	687
# twigs	1419	643	4321	612
% twigs	85	90	91	89

number of children a single hierarchy node has and the mean number of children per interior node respectively.

Unsurprisingly, the differences among different hierarchy types are much greater than the differences among different collections' hierarchies of the same type. Recent research by Murtagh [Murtagh 84b] and Griffiths and Willett [Griffiths & Willett 84] suggests that the single link hierarchy is

Table 2.5: Group average link hierarchy statistics

	CACM	CISI	INSPEC	MED
depth	43	40	52	27
max children	14	4	7	3
mean children	2.05	2.01	2.00	2.00
# interior nodes	3050	1446	12642	1027
# twigs	2045	907	8323	665
% twigs	67	63	66	65

Table 2.6: Complete link hierarchy statistics

	CACM	CISI	INSPEC	MED
depth	20	8	16	9
max children	327	115	802	78
mean children	2.17	2.08	2.07	2.08
# interior nodes	2729	1345	11878	956
# twigs	1928	650	7764	638
% twigs	71	63	65	67

significantly different from the hierarchies produced by other agglomerative, hierarchic clustering methods in terms of such characteristics as the skewness of the hierarchy (indicated by the depth of the hierarchy) and the size of the low-level clusters. These differences may be seen more easily in Figures 2.9-2.11. The Figures present the single link, group average link, and complete link hierarchies of a subset of the ADI collection. (This collection consists of 82 documents in documentation; it was not used in the experimental work due to its unrepresentative size.) To fit the hierarchies on a single page, they are represented as dendrograms instead of trees. Each splitting point is labeled with the similarity at which the cluster forms; the length of a line connecting two points is inversely proportional to the similarities it connects.

The lack of balance of the single link hierarchy is readily seen. Many clusters form simply by adding one more document to an existing cluster. Note in Table 2.4 that approximately 90% of the interior nodes are twigs. This behavior also causes the tree to be very deep and narrow.

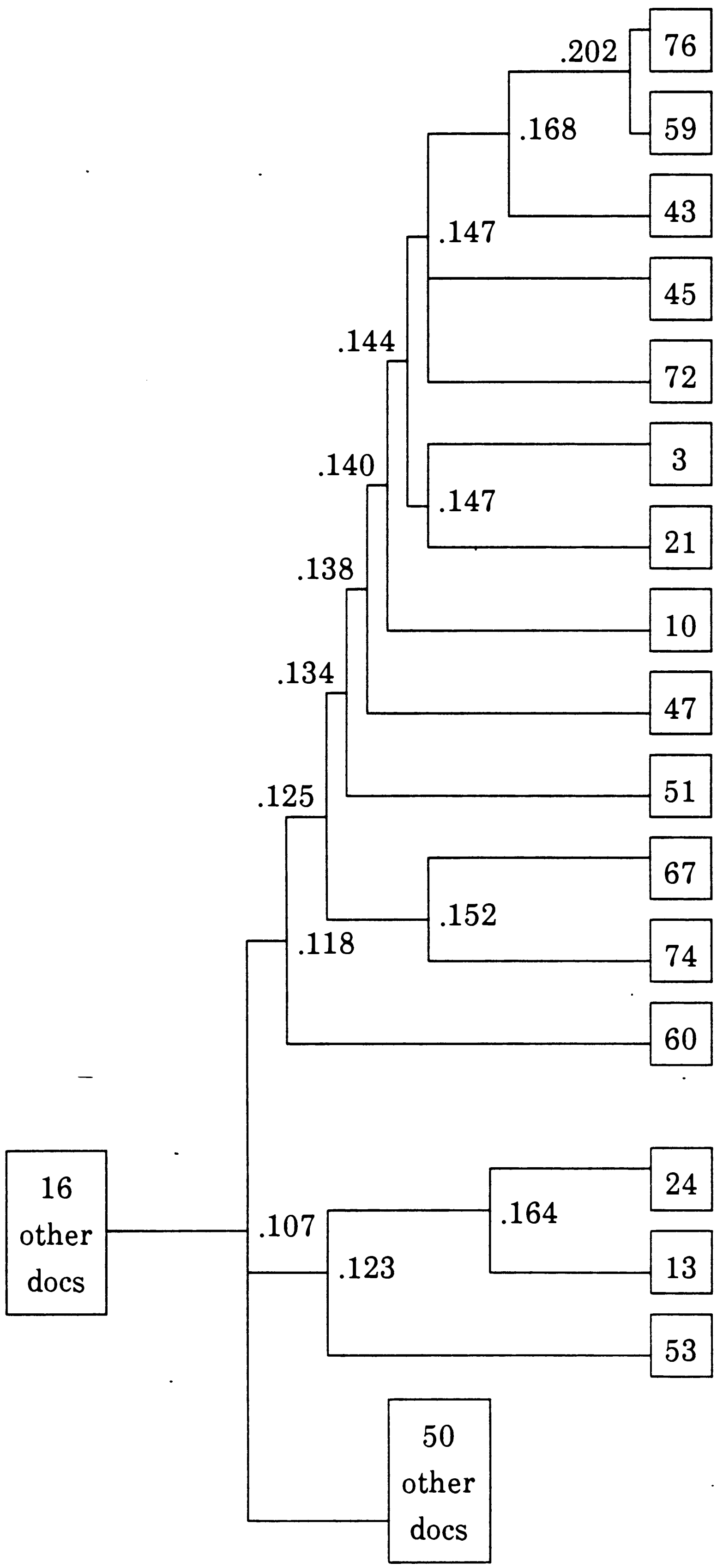


Figure 2.9: Single link hierarchy of the ADI collection

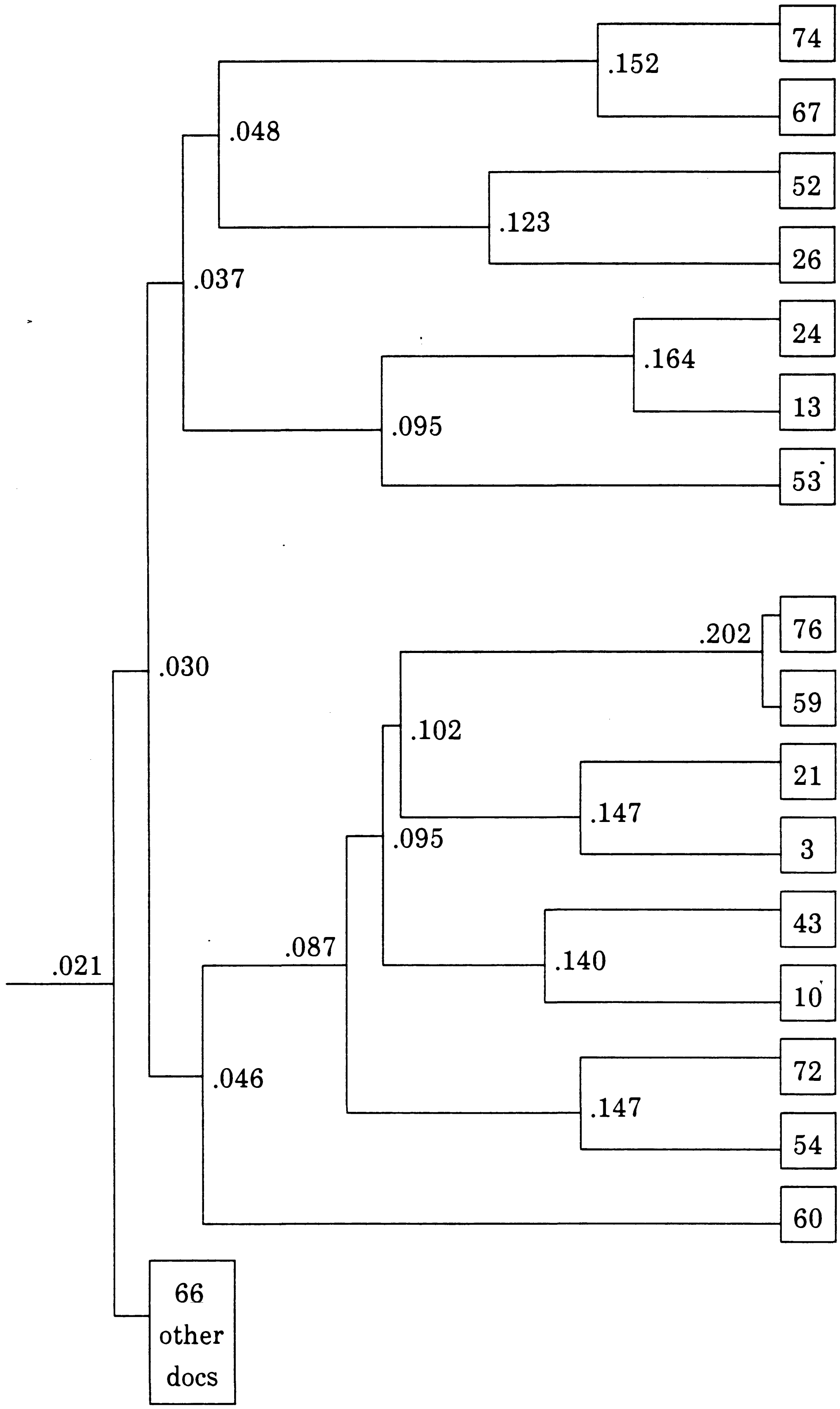


Figure 2.10: Group average link hierarchy of the ADI collection

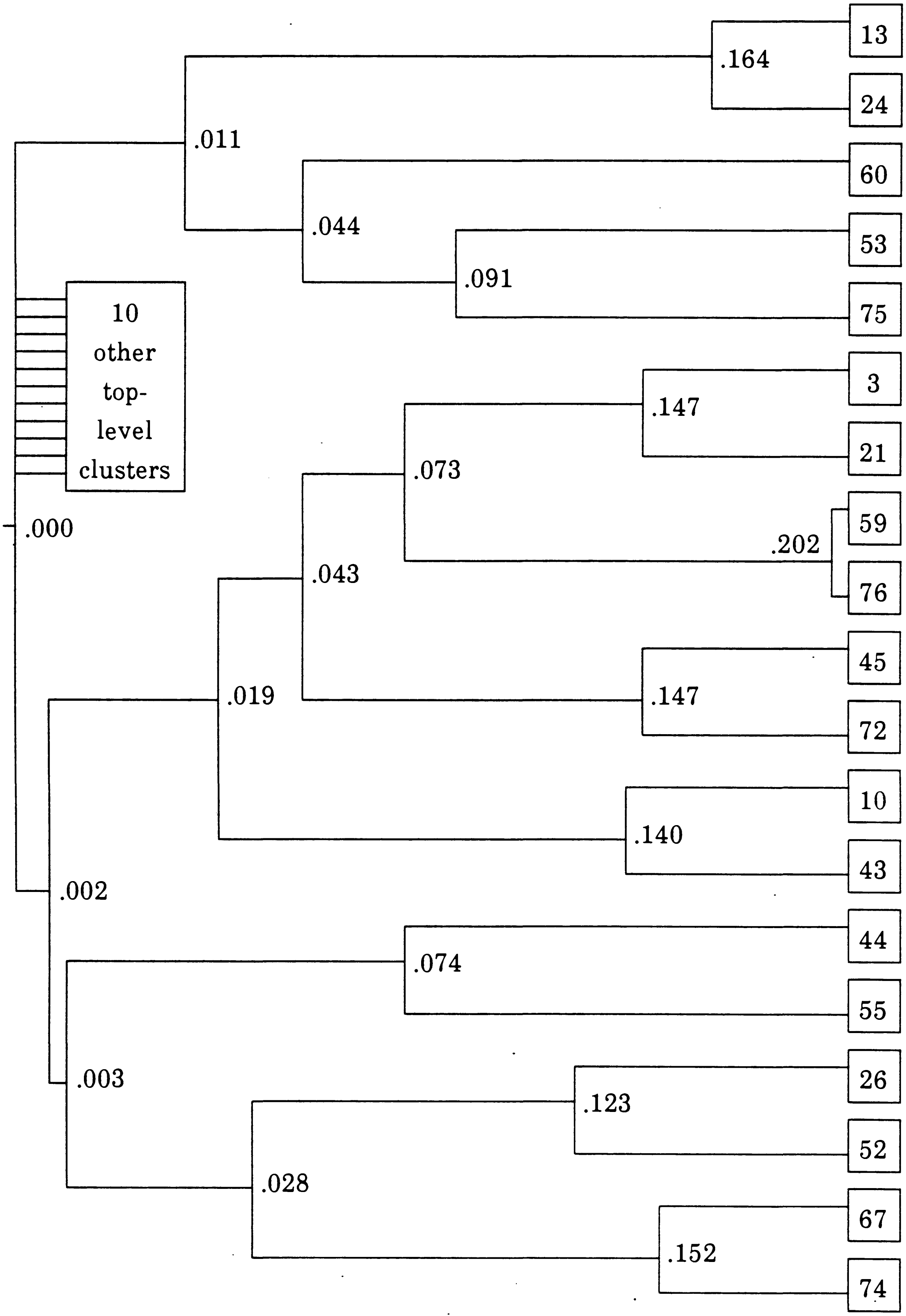


Figure 2.11: Complete link hierarchy of the ADI collection

The complete link hierarchy, on the other hand, is fairly well balanced and is much shallower than the other hierarchies. The shallowness is directly attributable to the high branching ratio of the root. (The clusters represented by the children of the root are called top-level clusters; in the complete link hierarchy, these top-level clusters merge at similarity level zero.) For each of the collections, the node with the maximum number of children as given in Table 2.6 is the root of the hierarchy. Although the mean number of children per interior node is approximately two for all the hierarchy types, this very large number of children of the root keeps the complete link hierarchy shallow.

The group average link hierarchy falls between the two extremes of the complete and single link hierarchies. It is neither as shallow as the complete link hierarchy nor as deep as the single link hierarchy. Similarly, it is neither as balanced as the complete link hierarchy nor as skewed as the single link hierarchy. In general, however, the characteristics of the group average link hierarchy are more similar to the characteristics of the complete link hierarchy than to the characteristics of the single link hierarchy.

Chapters 3 and 4 explore how the characteristics of a hierarchy, particularly its depth, affect the effectiveness and efficiency of the cluster searches.

Chapter 3
The Effectiveness of Cluster Searches

It has already been mentioned that the purpose of this thesis is to compare the effectiveness and efficiency of inverted index and cluster searching. The problem is not fully specified, however, since there is no generally agreed upon best cluster search. In the previous chapter the two principal variables that are to be tested were introduced: hierarchy type and searching strategy. These twelve candidates (three hierarchy types times four search strategies) for best cluster search must be supplemented by other parameters such as the number of documents to be retrieved and the maximum centroid length. These other variables may also affect the performance of the cluster searches.

The purpose of this chapter is to discover, through experimentation, which are the most effective cluster retrieval strategies. The effectiveness of ranking the entire document collection by decreasing similarity to the query (as produced by the inverted index search) is included in the comparisons. For the most part, consideration of the efficiency of clustered retrieval is postponed until Chapter 4 since a retrieval strategy must be minimally effective to be useful. At the same time, strategies which are clearly very expensive are not pursued in depth.

The experiments performed in this chapter demonstrate that clustered retrieval can be more effective than a full ranking of the document collection. Specifically, the top-down search which retrieves individual documents from within clusters of the complete link hierarchy is more effective than the inverted index search for all four collections. Furthermore, the evidence suggests that this search is likely to remain effective as the collection size increases.

The results of the experiments exhibit a wide variability in the effectiveness of the cluster searches. Top-down searches of the complete link hierarchy are more effective than the top-down searches of the single link hierarchy and, to a lesser extent, of the group average link hierarchy. This appears to be directly attributable to the differences in the depths of the hierarchies. For bottom-up searches, the group average link and complete link hierarchies produce similar results. These findings reinforce Griffiths' and Willett's conclusion that single link clustering is less effective than other agglomerative, hierarchic methods [Griffiths & Willett 84].

Bottom-up searching is superior to top-down searching except when retrieving individual documents from within complete link clusters. Croft also found bottom-up searches to be more effective than top-down searches for the single link hierarchy [Croft 80]. However, none of the bottom-up searches are more effective than the inverted index search for all four collections.

The amount to which the cluster hypothesis characterizes a collection does not give enough information to predict how well cluster searches will in general perform compared to the inverted index search [Voorhees 85]. It does give some indication of how well searches that retrieve entire clusters work in comparison with searches that retrieve individual documents from within clusters: if many of the documents that are similar to one another are also relevant to the same queries, the entire cluster searches can be more effective than the individual document searches. As a rule, however, the individual document searches are (much) more effective than the entire cluster searches.

The remainder of the chapter describes the experiments which support the conclusions listed above. The first section describes a set of experiments using only the single link hierarchy. The second section discusses the effectiveness of the four major searching algorithms on the three hierarchy types. The final section summarizes the searches that will be used in the efficiency experiments of Chapter 4.

3.1. Single Link Experiments

The single link clustering method has several advantages over the other agglomerative hierarchic methods. It is the only method that meets all of Jardine's and Sibson's criteria of adequacy [Jardine & Sibson 68]. It is the most computationally efficient method to construct. It also is the method that has been used the most in previous information retrieval experiments, presumably due to the first two advantages. The first set of experiments, therefore, concentrate exclusively on single link clustering.

3.1.1. Parameter Selection

It was noted in the introduction to this chapter that the number of documents to be retrieved and a restriction on the maximum length of centroids might affect the performance of the searches, but good values for these parameters are not known. The purpose of the experiments described in this section is to investigate these parameters.

The single link searches used in previous work were most effective when precision was emphasized [Jardine & van Rijsbergen 71; Croft 80]. Furthermore, few searchers are interested in very high recall, but search, instead, for a few relevant documents [Barraclough 77]. Thus, two fairly small values, ten and twenty, are used for the *NumWanted* parameter in these experiments. Four values are used for the maximum length of a centroid: 50, 100, 250, and 500. These values are the value used for x in the centroid creation algorithm given in section 2.3.2. That is, if there are more then the maximum number of distinct terms in the union of the set of terms of each document in the cluster, only the terms with the highest frequency remain in the centroid.

The experiments consist of performing each of the searches with each of the values of the parameters on all collections. Tables 3.1-3.4 contain the results of the experiments. (A full explanation of the evaluation measures is given in section 2.4. For the case when $NumWanted=10$, the evaluation is performed after ten documents are retrieved; for the case when $NumWanted=20$, the evaluation is performed after both ten (top number) and twenty (bottom number) documents are retrieved. The three measures are mean precision, total number of relevant documents retrieved, and total number of queries retrieving no relevant documents, respectively.)

The results of the experiments in which the number of documents to be retrieved is varied (part a *vs.* part b in Tables 3.1-3.4) show that only the **td-indiv** search exhibits substantial differences in the precision of the retrieved set after ten documents are retrieved; for that search, retrieving twenty documents is more effective than retrieving ten documents. The difference in the total number of documents retrieved by a set of queries for the **td-indiv** search

Table 3.1: CACM effectiveness comparisons for number of documents wanted and maximum centroid length for all search strategies

max lgth	bu-entire			bu-indiv			td-entire			td-indiv		
	prec	tot. rel.	# q's no rel	prec	tot. rel.	# q's no rel	prec	tot. rel.	# q's no rel	prec	tot. rel.	# q's no rel
50	.154	77	25	.192	96	18	.034	17	43	.044	22	34
100	.174	87	23	.200	100	19	.044	22	37	.070	35	27
250	.166	83	25	.198	99	19	.068	34	32	.082	41	32
500	.166	83	25	.198	99	19	.088	44	26	.102	51	25

a) $NumWanted = 10$

max lgth	bu-entire			bu-indiv			td-entire			td-indiv		
	prec	tot. rel.	# q's no rel	prec	tot. rel.	# q's no rel	prec	tot. rel.	# q's no rel	prec	tot. rel.	# q's no rel
50	.152	76	25	.192	96	18	.034	17	43	.072	36	32
	.133	133	19	.131	131	18	.026	26	39	.049	49	31
100	.172	86	24	.200	100	19	.044	22	37	.100	50	24
	.140	140	20	.136	136	18	.033	33	35	.068	68	20
250	.164	82	26	.198	99	19	.068	34	32	.140	70	19
	.140	140	20	.134	134	18	.057	57	26	.086	86	18
500	.164	82	26	.198	99	19	.088	44	26	.150	75	18
	.140	140	20	.134	134	18	.068	68	23	.107	107	15

b) $NumWanted = 20$

Table 3.2: CISI effectiveness comparisons for number documents wanted and maximum centroid length for all search strategies

max lgth	bu-entire			bu-indiv			td-entire			td-indiv		
	prec	tot. rel.	# q's no rel	prec	tot. rel.	# q's no rel	prec	tot. rel.	# q's no rel	prec	tot. rel.	# q's no rel
50	.174	61	15	.254	89	12	.129	45	17	.177	62	13
100	.209	73	13	.266	93	9	.103	36	19	.186	65	12
250	.231	81	12	.263	92	13	.120	42	17	.174	61	15
500	.231	81	12	.263	92	13	.074	26	22	.111	39	17

a) $NumWanted = 10$

max lgth	bu-entire			bu-indiv			td-entire			td-indiv		
	prec	tot. rel.	# q's no rel	prec	tot. rel.	# q's no rel	prec	tot. rel.	# q's no rel	prec	tot. rel.	# q's no rel
50	.166 .149	58 104	16 13	.254 .177	89 124	12 11	.129 .126	45 88	17 10	.197 .177	69 124	12 10
100	.209 .154	73 108	13 12	.269 .191	94 134	9 8	.086 .096	30 67	19 12	.226 .191	79 134	10 6
250	.231 .180	81 126	12 11	.263 .180	92 126	13 11	.120 .131	42 92	17 9	.231 .186	81 130	10 9
500	.231 .179	81 125	12 11	.263 .180	92 126	13 11	.074 .113	26 79	22 10	.183 .141	64 99	12 7

b) $NumWanted = 20$

Table 3.3: INSPEC effectiveness comparisons for number of documents wanted and maximum centroid length for all search strategies

max lgth	bu-entire			bu-indiv			td-entire			td-indiv		
	prec	tot. rel.	# q's no rel	prec	tot. rel.	# q's no rel	prec	tot. rel.	# q's no rel	prec	tot. rel.	# q's no rel
50	.242	186	19	.282	217	16	.040	31	60	.052	40	59
100	.249	192	18	.284	219	15	.038	29	59	.046	35	53
250	.240	185	17	.274	211	16	.033	25	62	.036	28	60
500	.243	187	18	.275	212	16	.043	33	57	.023	18	64

a) $NumWanted = 10$

max lgth	bu-entire			bu-indiv			td-entire			td-indiv		
	prec	tot. rel.	# q's no rel	prec	tot. rel.	# q's no rel	prec	tot. rel.	# q's no rel	prec	tot. rel.	# q's no rel
50	.235 .174	181 268	20 16	.282 .188	217 289	16 14	.040 .040	31 61	60 55	.061 .047	47 72	57 55
100	.248 .181	191 279	18 14	.284 .187	219 288	15 14	.039 .038	29 59	59 50	.061 .041	47 63	57 52
250	.239 .177	184 273	17 14	.275 .179	212 276	16 14	.033 .040	25 60	62 50	.043 .031	33 48	57 53
500	.240 .176	185 271	18 15	.275 .179	212 276	16 14	.043 .036	33 55	57 53	.062 .037	48 57	47 45

b) $NumWanted = 20$

Table 3.4: MED effectiveness comparisons for number of documents wanted and maximum centroid length for all search strategies

max lgth	bu-entire			bu-indiv			td-entire			td-indiv		
	prec	tot. rel.	# q's no rel	prec	tot. rel.	# q's no rel	prec	tot. rel.	# q's no rel	prec	tot. rel.	# q's no rel
50	.497	149	3	.587	176	3	.227	68	17	.197	59	13
100	.547	164	3	.597	179	3	.303	91	14	.267	80	12
250	.547	164	3	.597	179	2	.387	116	7	.333	100	5
500	.577	173	2	.610	183	2	.357	107	5	.360	108	5

a) $NumWanted = 10$

max lgth	bu-entire			bu-indiv			td-entire			td-indiv		
	prec	tot. rel.	# q's no rel	prec	tot. rel.	# q's no rel	prec	tot. rel.	# q's no rel	prec	tot. rel.	# q's no rel
50	.530 .440	159 264	3 3	.583 .460	175 276	3 3	.190 .190	57 114	17 13	.237 .202	71 121	10 8
100	.533 .452	160 271	3 3	.593 .465	178 279	3 3	.237 .262	71 157	14 10	.357 .305	107 183	9 6
250	.550 .427	165 256	3 2	.603 .437	181 262	2 2	.357 .415	107 249	7 4	.427 .367	128 220	5 3
500	.577 .443	173 266	2 2	.607 .427	182 256	2 2	.330 .397	99 238	5 2	.494 .382	147 229	2 1

b) $NumWanted = 20$

is almost always greater than ten, and is frequently greater than twenty. The difference for the other three searches is frequently less than five.

The results of the experiments in which the maximum centroid length is varied (different rows of the same column in Tables 3.1-3.4) show no consistent dependency of effectiveness on centroid length. Different searches of the same collection rank the different centroid length searches differently, as do identical searches of different collections. For example, the CISI **bu-entire** search ranks the centroid lengths as 250=500>100>50, but the CISI **td-entire** search ranks them as 50>250>100>500. Furthermore, the MED **td-entire** search ranks the centroid lengths as 250>500>100>50.

3.1.1.1. Number of Documents to be Retrieved

The similarity of the results when ten documents are wanted to the results when twenty documents are wanted for three of the searching strategies can be explained by considering the definition of those searches. In the entire cluster searches, the only difference between a search that retrieves ten documents and a search that retrieves twenty documents arises when a given cluster is too big to be retrieved when $NumWanted = 10$ but small enough to be retrieved when $NumWanted = 20$. As a rule, when this occurs the $NumWanted = 10$ search just retrieves a subcluster of the too large cluster that is small enough, and thus the search results are similar. Likewise, the only difference for the **bu-indiv** search is when the size of the low-level cluster of a document with a high similarity to the query is too big for the $NumWanted = 10$ search but not for the $NumWanted = 20$ search. In that case, The $NumWanted = 10$ search scores documents in a small enough subcluster against the query, and, again, the results of the *NumWanted*=10 search are similar to the results of the *NumWanted*=20 search. For the **td-indiv** search, however, retrieving more documents forces the search to explore more of the hierarchy (nodes must be popped from the heap until more documents are popped) resulting in better retrieval for the *NumWanted*=20 case. Since there is not a big difference in the performance when ten or twenty documents are wanted for three of the searches, and there is a definite improvement in the precision of the **td-indiv** search when twenty documents are wanted, *NumWanted*=20 is used in the remainder of the experiments in the thesis.

3.1.1.2. Centroids

It is intuitively appealing that the effectiveness of a search should increase as the maximum length of the centroids increases: the more information that is available in the centroids, the more discriminating the searches should be. However, Murray showed that the effectiveness of a search does not increase linearly with the maximum centroid length [Murray 72]. His experiments demonstrated that many of the low frequency terms can be removed from the centroids without a serious loss in effectiveness. Indeed, the reason for investigating the maximum length is to see how short centroids can be (since shorter centroids are more efficient) before deterioration becomes severe.

The results of the current experiments, however, exhibit a much wider variability in the effectiveness of a given centroid length than is exhibited in Murray's work. Part of the variability of the results may be explained by the interaction between which terms are included in the centroid and the relative weights assigned to those terms.

The centroid creation algorithm is discussed in detail in section 2.3.2. In short, those terms with the maximum frequency within the cluster are included in the centroid and are assigned weights based on the rank values of their frequency. These weights are subsequently modified, first by multiplying by an inverse document frequency weight, and then by normalizing the weights so that the sum of the squares of the weights equals one.

Since the centroids are created using the most frequent terms, a longer centroid of a given cluster contains all the terms in a shorter centroid of that cluster. Due to the normalization of the weights, the weights of the terms that appear in both centroids are relatively smaller in the longer centroid than in the shorter centroid. The inverse document frequency component of the weights complicates the interaction further. The high frequency terms that are included in the centroid are precisely those terms that are assigned the lowest inverse document frequency weights. Thus a term with a relatively small frequency, which is thereby excluded from a shorter centroid, is assigned a relatively high inverse document frequency weight in the longer centroid.

The variability of the effectiveness of searches with varying maximum centroid lengths underscores the necessity of further research into a theory of centroid creation and weighting. Until such a theory is developed, some centroids must be used. From the experimental results, centroids of maximum length 50 seem to be too short, although there are some cases in which that length is preferred. Centroids of maximum length 500 seldom perform much differently than centroids of maximum length 250, and as they are quite expensive to use, they also can be excluded. The choice between a maximum length of 100 and 250 is a toss-up. For the rest of the experiments in this thesis, centroids of maximum length 250 are used.

3.1.2. Hierarchy Modifications

In addition to the comparisons of how the *NumWanted* and maximum centroid length parameters affect the searches, the experiments in the previous section highlight some of the other features of the cluster searches. Perhaps the most notable of these features is the startlingly poor performance of the top-down searches, especially on the larger collections. This poor performance is due in part to the *loose* or *orphan* documents (documents that are not very similar to any other documents and, therefore, don't cluster well) that join the hierarchy very close to the root and impede the progress of the top-down searches.

Jardine and van Rijsbergen also encountered problems with their top-down searches on unmodified single link hierarchies due to loose documents [Jardine & van Rijsbergen 71]. They suggested two ways of modifying the hierarchy to avoid these problems: truncating the hierarchy at a predetermined similarity level thereby effectively removing all the documents that join the hierarchy above that point from the collection; or truncating the hierarchy but forcing the documents that join above the truncation point into the best cluster (the cluster the document is most similar to) below the truncation point. In their experiments on a collection of 200 documents, they found forcing documents into the best clusters to be the more effective method, but as the experiments were run only on this one small collection, further experimentation is necessary.

Before experimenting with either modification strategy, it is necessary to choose a similarity level at which to truncate the hierarchy. This is unfortunate since it requires finding the value of yet another parameter; the percentage of documents that would be removed from the collection was used as a guideline when choosing this level. For the CISI and MED collections, more than 20% of the documents would be removed if the truncation similarity level is as high as .15, although less than 3% of the collection would be lost when the truncation level is .1. Therefore, the .1 level is used in the experiments. Since the entire INSPEC collection clusters at the .1 level, these experiments are not run on that collection. Table 3.5 contains the effectiveness evaluation of top-down searching of modified hierarchies.

The results of the modified hierarchy searches are disappointing. The effectiveness of the top-down searches is still quite poor. For the td-entire search, the truncated hierarchy must produce very similar results to that of the full hierarchy since that search cannot retrieve any cluster that contains more than *NumWanted* (plus a constant) documents. Forcing documents into the best cluster is slightly beneficial. For the **td-indiv** search, forcing documents into the best clusters is occasionally detrimental. Although different truncation levels might produce slightly different results, it is unlikely that the effectiveness of the searches would improve dramatically using these modifications.

While the single link clustering method has nice theoretical properties, the single link hierarchies are deep and narrow, and, therefore, difficult to search top-down. The centroids near the root of the hierarchy represent very many documents, and choosing among the centroids is difficult. Croft describes this

Table 3.5: Effectiveness of loose document modification for top-down searches

	td-entire			td-indiv		
	prec.	total relevant	# q's no rel.	prec.	total relevant	# q's no rel.
unmodified	.068 .057	34 57	32 26	.140 .086	70 86	19 18
truncated	.068 .057	34 57	32 26	.140 .087	70 87	19 17
forced	.070 .056	35 56	31 25	.138 .087	69 87	20 18

a) CACM collection

	td-entire			td-indiv		
	prec.	total relevant	# q's no rel.	prec.	total relevant	# q's no rel.
unmodified	.120 .131	42 92	17 9	.231 .186	81 130	10 9
truncated	.120 .131	42 92	17 9	.229 .184	80 129	10 9
forced	.120 .133	42 93	17 9	.234 .189	82 132	10 9

b) CISI collection

	td-entire			td-indiv		
	prec.	total relevant	# q's no rel.	prec.	total relevant	# q's no rel.
unmodified	.357 .415	107 249	7 4	.427 .367	128 220	5 3
truncated	.357 .415	107 249	7 4	.433 .372	130 223	6 4
forced	.377 .412	113 247	7 4	.413 .365	124 219	7 4

c) MED collection

effect, using a model of the uncertainty involved in choosing a path down the hierarchy [Croft 80]. He concludes that the top-down search has a very high degree of uncertainty. Although Croft's top-down search does not permit back-tracking, the same type of effect occurs with the top-down searches used in these experiments. Thus a more promising strategy to improve the top-down searches would be to modify the hierarchy to make it more easily searchable.

Two different methods of accomplishing this objective are investigated. The goal of both methods is to produce a shorter and bushier hierarchy, but the two methods use very different means of obtaining this objective. The first method produces what will be called the *fixed* hierarchy. In the fixed hierarchy, certain similarity levels are chosen *a priori* and the documents are clustered using only those levels. For example, assume documents A and B have similarity .5 and documents A and C have similarity .51. In the full single link hierarchy there would be a node representing the cluster {A,C} at similarity level .51 and another node representing the cluster {A,B,C} at similarity level .5. If the chosen similarity levels for the fixed hierarchy were 0.0, .1, .2, ... 1.0, there would be a single node representing the cluster {A,B,C} at level .5. The set of clusters in the fixed hierarchy is simply a subset of the clusters in the full hierarchy. There still is no method of controlling the relative sizes of clusters, but all the theoretical properties of the single link method are preserved.

The second method, the *equal* hierarchy method, is based on the observation that a balanced tree is less deep than a skewed tree having the same number of leaves. The equal hierarchy is constructed from the full single link hierarchy by combining nodes of the full hierarchy into a single node until a threshold on the number of children for a single node is reached. In this way, the branching ratio of each of the nodes in the equal hierarchy is approximately the same. The effect of the fixed hierarchy algorithm is also to combine full hierarchy nodes, but in the equal case there are no pre-set similarity levels so dense areas of the full hierarchy do not end up in a single huge cluster as is possible in the fixed case.

However, the equal hierarchy is unsatisfactory for several reasons. The set of clusters that are represented in the equal hierarchy is not necessarily a subset of the set of clusters in the full hierarchy. After combining several nodes to obtain enough children, the new node frequently has too many children and must be split. This splitting may result in splitting some of the clusters of the full hierarchy. Furthermore, the process of combining nodes is order dependent, so the theoretical properties of the single link method are not preserved. Most importantly, the equal hierarchy does not necessarily form a more shallow tree. In fact, when the INSPEC equal hierarchy is constructed using a target of five children per node, the equal hierarchy is twice as deep as the full single link hierarchy. This happens because most interior nodes of the equal hierarchy have four leaves and only one non-leaf as a child. Due to these disadvantages, the equal hierarchy is not investigated further.

The characteristics of the modified single link hierarchies of the four collections are given in Table 3.6. The fixed hierarchy's similarity levels are intervals of .005, starting at 0.0; the hierarchy is created by rounding the similarities between documents down to the nearest .005 and using the single

Table 3.6: Hierarchy statistics for full and fixed single link hierarchies

	CACM		CISI		INSPEC		MED	
	full	fixed	full	fixed	full	fixed	full	fixed
Depth	389	111	143	40	306	78	141	41
max children	17	62	20	68	79	350	11	44
mean children	2.9	3.4	3.0	3.6	3.7	4.1	2.5	3.1
# interior nodes	1169	1342	730	553	4742	4058	687	501
% twigs	85	86	90	92	91	93	89	93

link algorithm described in section 2.2.3. The effectiveness of searches of the fixed hierarchy is summarized in Table 3.7.

Searching the fixed hierarchy is usually marginally more effective than searching the full hierarchy for all searching strategies. The individual document searches are more effective because these searches consider a larger proportion of the collection when searching the fixed hierarchy than when searching the full hierarchy since the mean cluster size is larger. However, since there aren't as many clusters represented in the fixed hierarchy, the larger proportion of the collection is considered while fewer similarity computations are performed. The entire cluster searches are also more effective on the fixed hierarchy than on the full hierarchy. An explanation of this may be that artificial division of clusters is avoided due to the coarser distinctions made when the similarities are computed.

As a final experiment with hierarchy modification, the loose document modifications described earlier are used in conjunction with the structure modifications. The precision of the top-down searches for the various combinations of modification are included in Table 3.8. The results are similar to that of the full single link hierarchy: no loose document modification makes a large difference. For the fixed hierarchy, however, forcing documents into the best cluster does not hurt performance and usually improves it slightly.

In the remainder of this thesis, "the single link" hierarchy will refer specifically to the fixed single link hierarchy as described above. For top-down searches, loose documents will be forced into the best cluster remaining in the hierarchy. There are two reasons for preferring the fixed hierarchy to the full hierarchy: the effectiveness of the searches is slightly better with the fixed hierarchy and the efficiency of the searches is much better due to the reduced number of clusters represented in the fixed hierarchy. These advantages are obtained without sacrificing the soundness of the single link clustering method.

3.1.3. Summary

This section has investigated the effectiveness of clustered retrieval using the single link clustering method. An initial set of runs experimentally determined good values for the parameters of the searches. Twenty documents is used as the target number of documents to be retrieved. Centroids of maximum length 250 are used in the rest of thesis, although a theory of

Table 3.7: Effectiveness of structure modification of the single link hierarchy

	bu-entire			bu-indiv			td-entire			td-indiv		
	prec	tot. rel.	# q's no rel	prec	tot. rel.	# q's no rel	prec	tot. rel.	# q's no rel	prec	tot. rel.	# q's no rel
full	.164 .140	82 140	26 20	.198 .134	99 134	19 18	.068 .057	34 57	32 26	.140 .086	70 86	19 18
fixed	.166 .142	83 142	25 20	.200 .135	100 135	19 18	.084 .072	42 72	27 21	.142 .089	71 89	19 17

a) CACM collection

	bu-entire			bu-indiv			td-entire			td-indiv		
	prec	tot. rel.	# q's no rel	prec	tot. rel.	# q's no rel	prec	tot. rel.	# q's no rel	prec	tot. rel.	# q's no rel
full	.231 .180	81 126	12 11	.263 .180	92 126	13 11	.120 .131	42 92	17 9	.231 .186	81 130	10 9
fixed	.243 .183	85 128	13 11	.263 .184	92 129	13 11	.126 .137	44 96	17 11	.223 .189	78 132	10 8

b) CISI collection

	bu-entire			bu-indiv			td-entire			td-indiv		
	prec	tot. rel.	# q's no rel	prec	tot. rel.	# q's no rel	prec	tot. rel.	# q's no rel	prec	tot. rel.	# q's no rel
full	.239 .177	184 273	17 14	.275 .179	212 276	16 14	.033 .039	25 60	62 50	.043 .031	33 48	57 53
fixed	.242 .173	186 267	18 14	.274 .177	211 272	15 14	.039 .040	30 62	59 46	.056 .036	43 56	51 46

c) INSPEC collection

	bu-entire			bu-indiv			td-entire			td-indiv		
	prec	tot. rel.	# q's no rel	prec	tot. rel.	# q's no rel	prec	tot. rel.	# q's no rel	prec	tot. rel.	# q's no rel
full	.550 .427	165 256	3 2	.603 .437	181 262	2 2	.357 .415	107 249	7 4	.427 .367	128 220	5 3
fixed	.570 .432	171 259	2 2	.620 .468	186 281	2 2	.357 .377	107 226	7 5	.410 .353	123 212	5 3

d) MED collection

Table 3.8: Effectiveness of structure modification in conjunction with loose document modifications

	td-entire			td-indiv		
	none	truncated	forced	none	truncated	forced
full	.068 .057	.068 .057	.070 .056	.140 .086	.140 .087	.138 .087
fixed	.084 .072	.084 .072	.086 .071	.142 .089	.142 .089	.150 .092

a) CACM collection

	td-entire			td-indiv		
	none	truncated	forced	none	truncated	forced
full	.120 .131	.120 .131	.120 .133	.231 .186	.229 .184	.234 .189
fixed	.126 .137	.126 .137	.126 .139	.223 .189	.220 .187	.229 .191

b) CISI collection

	td-entire			td-indiv		
	none	truncated	forced	none	truncated	forced
full	.357 .415	.357 .415	.377 .412	.427 .367	.433 .372	.413 .365
fixed	.357 .377	.357 .377	.377 .402	.410 .353	.410 .357	.417 .362

c) MED collection

centroid creation and weighting needs to be developed before the use these centroids can be completely justified.

For each of the collections used, the effectiveness of the top-down searches is very poor in comparison to the effectiveness of the bottom-up searches. This result is in keeping with previous work by Croft [Croft 80]. Modifications to the hierarchy designed to improve the performance of the top-down searches are only marginally useful for improving the effectiveness of the searches, but the fixed hierarchy is more efficient to search than the full single link hierarchy. The next section investigates other types of clustering methods with the goal of finding a more effective cluster search strategy.

3.2. Clustered Retrieval with Different Hierarchy Types

The experiments described earlier in this chapter have set the stage for the main purpose of this chapter: to compare the four different searching strategies – **bu-entire**, **bu-indiv**, **td-entire**, and **td-indiv** – on the three different hierarchy types – single link, complete link, and group average link – with one another and the inverted index search. The comparison is performed using a set of retrieval experiments to be discussed in the next subsection.

3.2.1 The Experiments

The experiments consist of searching each of the hierarchies using each of the searching strategies. This set of experiments is repeated for each of the four collections. With this design, direct comparisons can be made between each of the variables. The results of the clustered and inverted index searches are summarized in Tables 3.9-3.12. (The **bu-indiv** search used in these experiments is slightly different from the algorithm described in 2.3.2 and used in the single link experiments above. In the single link experiments, the documents contained in the ten most similar low-level clusters were ranked by decreasing similarity to the query. However, closer examination revealed that the ten most similar low-level clusters frequently did not contain twenty distinct documents, artificially depressing the evaluation after twenty

Table 3.9: CACM effectiveness comparisons for all search strategies and hierarchy types

search	(fixed) single link			complete link			group average link		
	prec	tot. rel.	# q's no rel	prec	tot. rel.	# q's no rel	prec	tot. rel.	# q's no rel
bu-entire	.166 .142	83 142	25 20	.206 .157	103 157	15 13	.198 .155	99 155	17 12
bu-indiv	.190 .160	95 160	18 15	.260 .207	130 207	11 10	.264 .223	132 223	10 9
td-entire	.086 .071	43 71	27 21	.134 .134	67 134	22 15	.094 .099	47 99	29 23
td-indiv	.150 .092	75 92	18 16	.286 .232	143 232	10 4	.264 .224	132 224	13 9

a) cluster searches

search	precision	total relevant	# q's no rel
inverted index	.256 .225	128 225	11 5

b) inverted index search

search	single link	complete link	group average link
bu-entire	−35.2	−19.5	−22.7
bu-indiv	−25.8	+1.6	+3.1
td-entire	−66.4	−47.7	−63.3
td-indiv	−41.4	+11.7	+3.1

c) percentage difference in precision over inverted index search after ten documents retrieved

Table 3.10: CISI effectiveness comparisons for all search strategies and hierarchy types

search	(fixed) single link			complete link			group average link		
	prec	tot. rel.	# q's no rel	prec	tot. rel.	# q's no rel	prec	tot. rel.	# q's no rel
bu-entire	.243 .183	85 128	13 11	.231 .189	81 132	13 10	.214 .186	75 130	13 9
bu-indiv	.260 .199	91 139	13 11	.271 .210	95 147	11 8	.280 .211	98 148	11 9
td-entire	.120 .139	44 97	17 10	.071 .100	25 70	20 11	.146 .147	51 103	21 12
td-indiv	.229 .191	80 134	10 8	.286 .256	100 179	11 4	.271 .243	95 170	9 6

a) cluster searches

search	prec	total relevant	# q's no rel
inverted index	.254 .244	89 171	9 5

b) inverted index search

search	single link	complete link	group average link
bu-entire	−4.3	−9.1	−15.7
bu-indiv	+2.4	+6.7	+10.2
td-entire	−52.8	−72.0	−42.5
td-indiv	−9.8	+12.6	+6.7

c) percentage difference in precision over inverted index search after ten documents retrieved

Table 3.11: INSPEC effectiveness comparisons for all search strategies and hierarchy types

search	(fixed) single link			complete link			group average link		
	prec	tot. rel.	# q's no rel	prec	tot. rel.	# q's no rel	prec	tot. rel.	# q's no rel
bu-entire	.242 .173	186 267	18 14	.287 .216	221 332	15 9	.277 .208	213 320	16 12
bu-indiv	.279 .205	215 315	15 13	.343 .255	264 393	12 8	.339 .253	261 389	11 8
td-entire	.039 .040	30 62	59 46	.160 .153	123 235	34 23	.078 .079	60 121	57 47
td-indiv	.056 .036	43 56	51 46	.364 .288	280 443	11 6	.253 .192	195 295	29 22

a) cluster searches

search	prec	tot. rel.	# q's no rel
inverted index	.348 .294	268 452	13 8

b) inverted index search

search	single link	complete link	group average link
bu-entire	−30.5	−17.5	−20.4
bu-indiv	−19.8	−1.4	−2.6
td-entire	−88.8	−54.0	−77.6
td-indiv	−83.9	+4.6	−27.3

c) percentage difference in precision over inverted index search after ten documents retrieved

Table 3.12: MED effectiveness comparisons for all search strategies and hierarchy types

search	(fixed) single link			complete link			group average link		
	prec	tot. rel.	# q's no rel	prec	tot. rel.	# q's no rel	prec	tot. rel.	# q's no rel
bu-entire	.570 .432	171 259	2 2	.583 .473	175 284	1 1	.603 .473	181 284	1 1
bu-indiv	.597 .490	179 294	2 2	.593 .533	178 320	2 2	.607 .552	182 331	1 1
td-entire	.377 .402	113 241	5 4	.547 .518	164 311	2 1	.493 .527	148 316	6 3
td-indiv	.417 .362	125 217	5 2	.600 .553	180 332	1 0	.590 .540	177 324	3 3

a) cluster searches

search	prec	tot. rel.	# q's no rel
inverted index	.577 .518	173 311	1 0

b) inverted index search

search	single link	complete link	group average link
bu-entire	−1.2	+1.0	+4.5
bu-indiv	+3.5	+2.8	+5.2
td-entire	−34.7	−5.2	−14.6
td-indiv	−27.7	+4.0	+2.3

c) percentage difference in precision over inverted index search after ten documents retrieved

documents retrieved. In these experiments, if the ten most similar low-level clusters contain at least twenty distinct documents, those documents are ranked. Otherwise, more low-level clusters are expanded, one at a time, until at least twenty distinct documents are in the union of the expanded clusters. All the documents in the union are then ranked.)

3.2.2. Average Effectiveness

The variability in the effectiveness of the cluster searches is apparent in the results of these experiments. The performance ranges from the very poor effectiveness of the top-down searches of the single link hierarchy (an 88% decrease in the precision of the inverted index search for the **td-entire** search of the INSPEC single link hierarchy) to the generally effective individual document searches of the complete link hierarchy (a 12% increase in precision over the inverted index search for the **td-indiv** search of the CISI complete link hierarchy).

Relative to the inverted index search, the clustered searches of the INSPEC collection perform worse than the other collections. For INSPEC, only one search, the **td-indiv** search of the complete link hierarchy, was more effective than the inverted index search. (The search exhibits an increase of 4.6% in precision after ten documents are retrieved; all percentages given in this section are percentage change over the inverted index search after ten documents are retrieved.) The bottom-up, individual document search of the complete link and group average link hierarchies of the INSPEC collection exhibit only minor degradation (−1.4% and −2.6%, respectively). The other INSPEC cluster searches show serious degradation (a loss of at least 17.5%).

The single link hierarchy is the hierarchy that produces the poorest results. The entire cluster and top-down searches of the single link hierarchy are less effective than the inverted index search for all four collections. The **bu-indiv** search of the CISI and MED single link hierarchies show small improvement (+2.4% and +3.5%, respectively).

The variability in the cluster searches is not confined to the average effectiveness of the searches. The effectiveness of individual queries also varies greatly in the clustered searches. Some queries are improved by the cluster searches while others are hurt. Note, for example, the performance of the **td-indiv** search of the group average link hierarchy for the CACM collection given in Table 3.9. After ten documents are retrieved, the average effectiveness of that search is greater than the average effectiveness of the inverted index search even though more queries retrieved no relevant documents; thus other queries necessarily retrieved more relevant documents in the cluster search than in the inverted index search. Other examples of this behavior can be found in the results of the other collections.

3.2.3. Hierarchy Type

The difference in the effectiveness of the searches on different hierarchy types was already mentioned. Searching the single link hierarchy of a collection is less effective than searching either the complete link or group average link hierarchies. The difference in effectiveness of the top-down searches is very pronounced. The effectiveness of the searches of the group average link hierarchy is more similar to that of the searches on the complete link hierarchy, although the complete link hierarchy top-down searches are more effective.

These results confirm the conclusion reached by Griffiths and Willett [Griffiths &Willett 84]. They investigated the effectiveness of a cluster based, bottom-up search on four hierarchy types (the three hierarchy types used in this thesis and the Ward method hierarchy), and concluded that the single link hierarchy is not necessarily the clustering algorithm of choice for information retrieval. This conclusion was based on the examination of the structure of the hierarchies and the results of retrospective retrieval experiments. The examination of the structure of the hierarchies revealed that the single link hierarchy contains larger low-level clusters than the other hierarchies contain. For their search, this implies that many of the documents in the single link hierarchy are not retrievable since a limit on the maximum size of a cluster that can be retrieved is imposed. (This effect is mitigated somewhat in the bottom-up searches used in this thesis since small enough children of too large clusters can be retrieved.) The retrospective retrieval experiments showed that the maximum effectiveness obtained by retrieving the single best single link cluster in response to each query was significantly less than the maximum effectiveness obtained using the other hierarchy types.

The predictive experiments performed by Griffiths and Willett showed few significant differences in the effectiveness of the searches of different hierarchy types, however. The failure of their search to discriminate among hierarchy types may be attributed to two factors. One, the (only) search used was a bottom-up search; the difference in the effectiveness of bottom-up searches of different hierarchy types is smaller than the difference in the effectiveness of the top-down searches. More importantly, the collections used by Griffiths and Willett were all smaller than 1000 documents. The results of the current experiments show a definite dependency on the size of the collection: the difference in the effectiveness of the top-down searches of all three different hierarchy types and the difference in the effectiveness of the bottom-up searches of the single link hierarchy versus the other two hierarchy types are larger on the larger collections. As collections in operational systems are at least as large as the INSPEC collection, and usually many times larger, this dependence is important.

The difference in the effectiveness of the different hierarchies for the bottom-up searches is explained by the difference in the sizes of the low-level clusters. The group average link and complete link hierarchies have similar low-level clusters and thus similar bottom-up performance.

The evidence suggests that the difference for the top-down searches is due to the difference in the depths of the hierarchies. A more shallow hierarchy implies that the largest clusters in the hierarchy (ignoring the degenerate cluster of the entire collection) are, in fact, not very large. This in turn implies that the centroids of the clusters can do a more accurate job of representing the clusters. There is other evidence which suggests hierarchies with smaller maximum cluster sizes are more effective to search. Previous SMART experiments supported the conclusion that the use of many small clusters was more effective than the use of a few large clusters [Grauer & Messier 71]. The slight improvement in the effectiveness of the fixed single link hierarchy over the effectiveness of the full single link hierarchy also supports this conclusion.

The dependence of the effectiveness of top-down searches on the size of the largest clusters and consequently on the depth of the hierarchy can also explain why the difference among hierarchy types becomes more apparent as collections get larger. The number of top-level clusters in the complete link hierarchy appears to grow linearly with the size of the collection (see Table 2.6). Complete link clusters get more and more difficult to join as they get larger since, by definition, a document must have at least the minimum similarity with every other document in the cluster in order to join the cluster. As a result, the size of the top-level clusters remains (relatively) constant independently of collection size for this hierarchy type. Therefore, the top-down search of the complete link hierarchy is likely to remain effective as the collection size increases. For the other hierarchy types, the maximum size of a cluster grows as the collection size grows; thus more uncertainty is introduced in the early (and most important) stages of the top-down searches.

3.2.4 Searches

The remaining variable that needs to be discussed is the searching strategy itself. Various aspects of the searches have already been mentioned. For example, it has already been noted that the bottom-up searches are less affected by hierarchy type than the top-down searches, and that the most effective search is the **td-indiv** search on the complete link hierarchy. The conclusion that a top-down search is more effective than the corresponding bottom-up search is in opposition to Croft's conclusion discussed in section 3.1.2. However, the top-down search is more effective only for the complete link hierarchy. For the single link and group average link hierarchies, the bottom-up search is more effective than the top-down search for these four collections.

Regardless of whether the search is top-down or bottom-up, retrieving individual documents from within clusters is more effective than retrieving entire clusters in response to a query. The only collection for which this is not consistently true is the MED collection – the collection which is characterized by the cluster hypothesis to the greatest extent (see Table 2.3). For this collection, the effectiveness of some of the bottom-up entire cluster searches of the complete link and group average link hierarchies is very similar to the effectiveness of the bottom-up individual document searches.

Two conclusions may be drawn from this observation. First, a collection must be strongly characterized by the cluster hypothesis if the retrieval of entire clusters is to be an effective selection strategy. It is unlikely that the collections used in operational systems are characterized by the cluster hypothesis to the extent necessary for making retrieval based on entire clusters worthwhile. The MED collection was originally created by selecting documents out of a larger database available at the National Library of Medicine. The documents in the collection are either documents that were retrieved in response to a query in an initial search at the Library (source documents), or documents that were cited by a source document. As a consequence of constructing the collection in this manner, most of the documents in the collection are included as a result of the document's relationship to some particular query. This characteristic of the MED collection makes it more likely that documents that are similar to one another are relevant to the same queries than if a more general method of constructing the collection were used. Since MED is the only one of the four collections used that is sufficiently characterized by the cluster hypothesis to make retrieving entire clusters competitive with the individual document searches, the general usefulness of cluster based retrieval is in doubt.

Note, however, that not retrieving entire clusters does not mean that document clustering is not useful. Searching strategies that retrieve individual documents from within clusters based on the documents' similarity to the query can improve upon the effectiveness of a total ranking of the document collection. Indeed, it has already been shown that the top-down search that retrieves individual documents from within complete link clusters is more effective for high precision searches than a full ranking of the document collection. The improvement results from the cluster search not encountering some nonrelevant documents that have a high similarity to the query due to the documents being in unexpanded clusters. In this case, the knowledge inherent in the hierarchy – that the nonrelevant document is similar to documents that are not similar to the query – successfully prevents the retrieval of a nonrelevant document.

The second conclusion is that the single link hierarchy does not form tight enough clusters to make retrieving entire clusters worthwhile even if the cluster hypothesis is true for a collection. The scraggly clusters formed by the single link method cause documents that are similar to one another to be fairly far apart in that hierarchy. Thus there is seldom a single, well-defined cluster to be retrieved in response to a query.

3.3. Conclusion

The effectiveness of agglomerative hierarchic clustering methods was investigated in this chapter through a series of retrieval experiments. The retrieval experiments both verified some tentative conclusions made by earlier researchers by showing the same conclusions hold for different, larger collections and presented new results.

- For the four collections tested, the top-down search that retrieves individual documents from complete link clusters was more effective than a total ranking of the document collection by decreasing similarity to the query. How much improvement is obtainable seems to be independent of how well the cluster hypothesis characterizes a collection: the maximum improvement of a cluster search over the full ranking for MED, the collection that is most clearly characterized by the cluster hypothesis, is 5.2% while for CISI, the collection that is clearly not characterized by the hypothesis, the maximum improvement is 12.6%.

- Cluster based retrieval (retrieving entire clusters in response to a query) was less effective than retrieving individual documents from within clusters. This is a consequence of how poorly the cluster hypothesis characterizes most collections.

- Griffiths' and Willett's conclusion that the single link hierarchy is less effective for searching than the hierarchies produced by other agglomerative hierarchic methods [Griffiths & Willett 84] is substantiated. Whereas Griffiths and Willett based their conclusion on studies of the hierarchies' structures and retrospective experiments, these experiments demonstrated that predictive searches are also more effective on other hierarchy types.

- Croft's model of the uncertainty involved in top-down searches [Croft 80] is accurate for the group average link and single link hierarchies. However, the depth of the complete link hierarchy is so restricted that top-down searching is more effective than bottom-up searching on that hierarchy. Since the depth of the complete link hierarchy appears to be independent of collection size, it is likely that the top-down search will remain effective for larger collections.

- The effectiveness of clustered retrieval exhibits a wide variability. One of the reasons the class of agglomerative hierarchic methods was chosen for investigation is that the clustering algorithms do not depend on the values of user-supplied parameters to control the shape of the hierarchy. Nonetheless, several parameters needed to be experimentally determined. The variability of the cluster searches was evident not only in the large difference in performance caused by changes to obviously important variables such as hierarchy type, but also in the unpredictability of the effect caused by changes to apparently minor parameters such as the precision to which the similarity between documents is computed.

The next chapter contrasts the efficiency of the inverted index and effective cluster searches. In particular, the efficiency of the individual document searches of all the hierarchy types will be investigated. The entire cluster

The third section describes the results of the processing time experiments. The top-down search of the single link and group average link hierarchies is the most efficient search asymptotically, but the search of those hierarchies is more efficient only when it is less effective than the inverted index search (on very large collections). For collections of a given size, the relative efficiency of the inverted index search and the top-down cluster search depends on the number of similarities that can be computed in the time it takes to access a page, for the time of an inverted index search is dominated by the CPU time while the time of a top-down search is dominated by the I/O activity. The bottom-up search is a hybrid search; it performs more page accesses than the inverted index search but fewer than the top-down search, and it computes fewer similarities than the inverted index search but more than the top-down search. When the terms in the queries appear in few documents on average, the inverted index search is the most efficient search; when this is not true – either because of the particular characteristics of the collection or because of the size of the collection – the top-down search is more efficient than the inverted index search. However, in both cases the inverted index search is efficient relative to a cluster search when an inexpensive similarity measure is used.

The final section summarizes the efficiency experiments. The conclusions reached differ from those of Murray who performed similar comparisons [Murray 72]. An explanation of the differences is given in that section.

4.1. Experimental Methodology

In order to compare the effectiveness of the searches, assumptions about the implementations of the searches must be made. The methodology used in the search of the single link hierarchy, the loose documents are forced into the best clusters below the truncation point.

The efficiency of the searches is measured by the mean time required to process a query and the amount of storage space required for the auxiliary data structures (the cluster hierarchy and centroids or the inverted index). The cost of originally creating these data structures – which is greater for clustering than for the inverted index – is not included in the comparison. Creating the appropriate data structures needs to be done infrequently compared to the rate of searching (for static collections, it needs to be done only once) and it does not have to be done while a user is waiting for a response. Furthermore, if the cluster searches prove to be efficient, additional research may develop methods of inexpensively approximating the cluster hierarchies. For example, Willett has shown that the single link hierarchy created by considering only the similarities between nearest neighbors is as effective as the single link hierarchy created by considering all inter-document similarities [Willett 84].

The design of the efficiency experiments is discussed in the first section of the chapter. Details of the assumptions made about the implementation of the data structures are also included in that section.

The second section compares the storage requirements of the cluster and inverted index searches. For all of the collections, the inverted index search requires less space than searches of the complete link or group average link hierarchies and approximately the same amount of storage as searches of the single link hierarchy. The larger space requirements for the cluster searches are due to the number and size of the centroids that need to be stored.

searches will not be considered since the effectiveness of those searches is poor in general. Although the single link searches are usually less effective than the searches of the other hierarchy types, the efficiency of those searches will be measured for completeness.

Chapter 4
Search Efficiency

Document clustering was originally conceived of as a method to improve the efficiency of searching a collection. The SMART project used clustering as an alternative to a sequential scan of the collection [Salton 71b]. Jardine and van Rijsbergen also advocated the use of document clustering for its fast search time as compared to a sequential scan [Jardine & van Rijsbergen 71]. However, the availability in recent years of computers with large amounts of internal storage has made it feasible to use other procedures for finding the most similar documents to a query without performing a sequential scan of the collection. These procedures use an inverted index of the collection to avoid processing documents without terms in common with the query [Smeaton & van Rijsbergen 81]. These inverted file searches are also much more efficient than complete sequential scans, and may obviate the use of document clustering solely as an efficiency enhancement.

The purpose of this chapter is to compare the efficiency of some of the more effective cluster searches with the efficiency of the inverted index search. The cluster searches that are used in the comparisons are the top-down and bottom-up searches that retrieve individual documents. The entire cluster searches are necessarily more efficient than the individual document searches, but the effectiveness results of the previous chapter suggest that these searches are too ineffective to be of practical use regardless of their efficiency. As in the previous experiments, the fixed single link hierarchy is used; for the top-down efficiency experiments, including the assumptions that are made, is described in this section.

4.1.1. Processing Time

The more important of the two components of the efficiency of a searching strategy is the mean amount of time it takes to retrieve a set of documents for a query. The total time 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; it encompasses such costs as the floating point arithmetic performed in the similarity computations, determining which query terms are in a vector, and the overhead involved with subroutine calls. The I/O time is the amount of time the search spends getting pages from secondary store (I assume the secondary storage device is a disk).

Computing the similarity between a p-norm query and a vector (either a centroid or a document) entails traversing the tree that represents the query exactly once. The cost in terms of the number of floating point operations, recursive calls, *etc.* is the same for each traversal regardless of the number of terms the vector has in common with the query. As the CPU time of a search is dominated by the number of query-vector similarities computed in the search, and the cost of an individual similarity computation is independent of the searching strategy used, the number of query-vector similarities computed is used as the measure of the CPU cost of a search. The inverted index search computes one similarity for each document in the union (no duplicates) of the inverted lists of the query terms (see section 2.3.1). The cluster searches compute the similarity of the query with each centroid and document the search encounters in the hierarchy traversal.

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. Similarly, the bottom-up search needs only the inverted lists of the low-level centroids of the query terms, and then the vectors of the documents in the selected clusters. Since these inverted centroid lists are no longer required once the query-document similarities begin to be computed, the document vectors can overwrite the inverted lists; furthermore, since a document vector is not needed after its similarity is computed, the vectors can each be read into the same memory location.

The top-down search needs to keep the hierarchy nodes that have been visited and the current centroid or document vector in core. Again, each vector is needed only once (if it is needed at all) so the same memory location can be used for each vector. The hierarchy nodes may be accessed more than once, so they must be stored in separate locations. Since each node consists of a small number of bytes (24 bytes is assumed in the experiments later in this chapter) 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).

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 memory usage considerations above do not include the memory needed by the program variables such as the array of similarities of the inverted index search and the heap used to store the similarities in the top-down cluster search. The amount of space needed by the program variables is negligible and is not included in the efficiency experiments.)

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 processing time experiments do not use this level of detail.

4.1.2. Data Structure Configuration

Counting the number of pages that need to be accessed to perform a search implies the configuration of the data is known. This section describes how the data is allocated to pages for these experiments.

The model of page allocation used 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 were added to the collection. The page length is assumed to be 4096 bytes.

The inverted file consists of a set of list headers plus the lists associated with each header; it contains an entry for each distinct concept that occurs in the inverted vectors. A header consists of the concept number, the list length, and a notation of the page on which the list begins (12 bytes). A list contains one tuple for each vector in which the corresponding concept appears. Each tuple contains the vector id and the weight of the concept in that vector (8 bytes). 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.

The hierarchy node representing a cluster is of fixed length; it contains the node id, the similarity level at which the cluster it represents was formed, a notation of the page on which its associated vector begins, the number of children, the node id of its first child, and the node id of its rightmost sibling (24 bytes). The children of a given node are stored together beginning on a new page. They take as few pages as possible without splitting a node across page boundaries. 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.

The page on which an inverted index header or hierarchy node resides is computable from the concept number or node id. For example, since both the headers and the hierarchy nodes are of fixed length, the page may be calculated using the byte offset from the beginning of the file. Thus the header and hierarchy file are immediately accessible.

Like inverted lists, a vector also has one fixed length and one varying length component, although for vectors the two parts are stored consecutively. The fixed length header consists of the id number and number of tuples in the variable length part (8 bytes). Each tuple consists of a concept number and weight of the concept in the vector (8 bytes). A vector is started on a new page and takes as few pages as possible without splitting a tuple across page boundaries. Access to a vector (a notation of the page on which the vector begins) is obtained through the hierarchy node associated with the vector.

With these configurations for the data structures, the number of pages required by an inverted index search is the number of distinct pages which contain query term headers plus the lengths (in pages) of the inverted lists for the query terms. The bottom-up search accesses the number of pages required for the inverted index search of the low-level centroids plus the number of distinct pages that contain the selected hierarchy nodes plus the length (in pages) of the document vectors in the selected clusters. The top-down search accesses the number of distinct pages that contain the hierarchy nodes encountered in the traversal of the hierarchy plus the length (in pages) of each of the vectors associated with the encountered nodes.

These page counts do not include the pages that need to be accessed in order to retrieve the natural language text returned to the user. The inverted index search will, on average, need to access a random page for each document returned to the user. Since the natural language text of the documents can be stored by cluster if the collection is clustered, the cluster searches will require fewer page accesses than the inverted index search in general. However, only a few documents (twenty in these experiments) are returned, so the difference between the number of pages accessed to retrieve the text of the documents is negligible compared to the difference between the number of pages accessed in the searches.

4.1.3. Disk Space Requirements

The second component of the efficiency of the searches is the amount of external memory required to store the data. As with the page count experiments, the storage required for the natural language text of the documents will not be included in the comparisons since the same amount of storage is required for the text regardless of the searching strategy used.

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 required by both the direct and inverted files. The space measured for the bottom-up search is the sum of the space required by the hierarchy file, the inverted file of the low-level centroids, and the direct document file. For the top-down search, the space measured is the space required by the hierarchy file, the direct centroid file, and the direct document file.

4.2. Disk Usage Experiments

The amount of external memory necessary to store a document collection and the data structures required to perform a search are investigated in this section. The size is measured by the number of bytes required to store the actual data since the number of pages required by the collection when configured as described above is not an accurate description of the relative sizes of the auxiliary data structures. For example, the configuration of the inverted index calls for each term list to begin on a new page. This configuration accurately models the I/O activity of a search since the terms are added to the inverted index randomly. In a large collection a different page will be accessed for each term on average. This is not an accurate picture of space requirements, though, since the terms that appear in the collection only once are allocated an entire page when they need only eight bytes. The bottom-up searches suffer the same penalty in the size of the low-level centroid inverted index.

The number of bytes of data required by the three searching strategies is given in Tables 4.1-4.3. The size of each of the data structures and the total over all the structures is given. For the cluster searches, the percentage change over the inverted index search is also included.

The space required by the inverted index search is smaller than the space required by any of the searches of the complete link or group average link

Table 4.1: Number of bytes required for inverted index search

	inverted index of documents	document vectors	total
CACM	679,892	603,488	1,283,380
CISI	572,084	539,584	1,111,668
INSPEC	3,472,916	3,399,512	6,872,428
MED	509,516	434,656	944,172

hierarchies, and is approximately equal to the space required by the single link searches. The space requirements of the top-down searches are larger than those of both the inverted index and the bottom-up searches. The cluster searches require larger amounts of disk space due to the number of centroids that need to be stored. For each collection, the centroids of the complete link and group average link hierarchies use more space than the inverted index of the documents. Even the inverted index of the low-level centroids is larger than the inverted index of the documents for these two hierarchy types. The single link hierarchy is the most space efficient of the hierarchies, mostly due to the fact that the the fixed single link hierarchy is used instead of the full single link hierarchy. The fixed single link hierarchy contains many fewer clusters than the full hierarchy, and, therefore, many fewer centroids need to be stored. The group average link hierarchy contains the largest number of centroids and is the least space efficient hierarchy.

Table 4.2: Number of bytes required for **bu-indiv** search and percentage change over inverted index search

		inverted low centroids	hierarchy nodes	document vectors	total	% change
CACM	single	477,520	109,104	603,488	1,190,112	−7
	complete	780,732	142,392	603,488	1,526,612	+19
	ave	944,468	150,096	603,488	1,698,052	+32
CISI	single	491,212	48,312	539,584	1,079,108	−3
	complete	642,748	67,320	539,584	1,249,652	+12
	ave	733,096	69,744	539,584	1,342,424	+21
INSPEC	single	2,431,580	401,808	3,399,512	6,232,900	−9
	complete	4,203,644	589,488	3,399,512	8,192,644	+19
	ave	5,255,456	607,824	3,399,512	9,262,792	+35
MED	single	621,024	36,816	434,656	1,092,496	+16
	complete	648,472	47,736	434,656	1,130,864	+20
	ave	715,056	49,440	434,656	1,199,152	+27

The amount of disk space required by the cluster searches would, of course, be reduced if centroids of maximum length 100 were used instead of maximum length 250 centroids. Assuming the size of the inverted index of low-level centroids would decrease by sixty percent (a false assumption as not all low-level centroids contain 250 terms), the bottom-up search of all hierarchy types would require less storage than the inverted document index search. Assuming the size of all centroids would decrease by sixty percent (a more reasonable assumption), the top-down search of the complete link and group average link hierarchies would still require approximately thirty percent more space than the inverted index search. If the direct document file is not counted as part of

Table 4.3: Number of bytes required for **td-indiv** search and percentage change over inverted index search

			hierarchy	document		%
		centroids	nodes	vectors	total	change
CACM	single	481,792	108,672	603,488	1,193,952	−7
	complete	1,481,320	142,392	603,488	2,227,200	+74
	ave	2,066,696	150,096	603,488	2,820,280	+120
CISI	single	514,768	48,192	539,584	1,102,544	−1
	complete	1,375,536	67,320	539,584	1,921,850	+73
	ave	1,597,192	69,744	539,584	2,206,520	+98
INSPEC	single	2,624,184	401,808	3,399,512	6,425,504	−7
	complete	9,685,704	589,488	3,399,512	13,674,704	+99
	ave	11,842,976	607,824	3,399,512	15,850,312	+131
MED	single	615,464	36,648	434,656	1,086,768	+15
	complete	1,132,544	47,736	434,656	1,614,936	+71
	ave	1,313,296	49,440	434,656	1.797,392	+90

the inverted index space requirements, the inverted index search would be more space efficient than any of the cluster search/ hierarchy combinations.

4.3. Processing Time Experiments

Secondary storage is relatively inexpensive. It is unlikely that any of the storage requirements of the searches discussed above would be prohibitively expensive for an operational retrieval service if there were other reasons to prefer the search. The amount of time necessary to perform a search is a much more critical factor of the usefulness of a search since the searches are done while users wait for the results.

4.3.1. Processing Time of Original Queries

In these experiments the total time required by a search was measured by the number of query-vector similarities computed (CPU time) plus the number of pages accessed (I/O time). For each query, the number of similarities computed between it and a centroid or document vector was counted. The number of pages accessed in the search for that query was determined by making a trace of the search. The trace contained a notation of which item was fetched every time an inverted index list, a hierarchy node, or a vector was needed. Assuming that the collection was configured as described in section 4.1.2, the number of pages that were accessed at least once could be computed using the trace. Each query was assumed to be independent of all other queries; that is, simulated memory was cleared after each query. The mean number of query-vector similarity computations computed over the query set for each collection is given in Table 4.4.

For the inverted index search hundreds (or thousands) more similarities than page accesses are computed (see column one of Table 4.4). This search needs to access only the pages that contain the inverted index entries of query terms. Since queries are usually short, the inverted index search is very efficient in the number of pages accessed. However, a query-vector similarity must be computed for each distinct document vector in the union of the query term entries. As query terms tend to be frequent terms, the number of similarities computed is relatively large. Thus both the CPU time and the I/O time of the

Table 4.4: Mean number of query-vector similarity computations and mean number of pages accessed per query for inverted index and individual document cluster searches

	inverted index	(fixed) single link		complete link		group average link	
		bu	td	bu	td	bu	td
mean sims	563.0	346.0	505.4	561.6	408.9	649.1	168.9
mean pages	8.7	81.3	528.8	68.3	452.5	73.9	247.6

a) CACM collection

	inverted index	(fixed) single link		complete link		group average link	
		bu	td	bu	td	bu	td
mean sims	913.9	460.6	663.6	716.3	224.5	777.7	183.9
mean pages	13.7	58.1	692.4	58.5	281.2	59.7	271.7

b) CISI collection

	inverted index	(fixed) single link		complete link		group average link	
		bu	td	bu	td	bu	td
mean sims	4463.4	1967.3	684.2	3871.0	873.0	4338.1	133.6
mean pages	24.0	69.8	700.4	73.1	914.5	78.6	195.6

c) INSPEC collection

	inverted index	(fixed) single link		complete link		group average link	
		bu	td	bu	td	bu	td
mean sims	367.7	312.3	308.2	345.7	144.5	367.6	85.6
mean pages	36.2	108.1	330.2	83.0	179.8	84.4	128.4

d) MED collection

inverted index search depend on the mean number of terms per query and the mean number of documents applicable to each query term. Since the mean number of documents per query term depends in turn on the mean length of a document vector and the number of documents in the collection, these factors indirectly influence the efficiency of the inverted index search.

The efficiency of the bottom-up cluster search also depends on the number and mean frequency of query terms since the most expensive part of the bottom-up search is the inverted index search of the low-level centroids. The inverted search of the low-level clusters must select the best clusters from a large number of low-level clusters. After the appropriate clusters have been located using the inverted search, only the twenty or so documents contained in those clusters need to be examined. (The cost of the search will increase slightly as *NumWanted* increases, however, since more clusters must be expanded which implies more pages must be accessed.) Once again, several hundred more similarity computations are performed than page accesses.

The efficiency of the top-down cluster search is (almost) independent of the query length and the frequency of the query terms. The efficiency of the top-down searches depends instead on the number of hierarchy nodes that are expanded. The number of nodes expanded in the search depends in turn on the number of documents to be retrieved, the hierarchy characteristics, and other query-dependent factors which control how much back-tracking the search must perform. Unlike the other two searches, the top-down search accesses more pages than it computes similarities: since a vector is stored on its own page(s), each similarity requires at least one page access to fetch the vector associated with the current node. The search must also accesses the page(s) that contain the hierarchy nodes. However, the search is able to avoid processing many of the document vectors, so it is relatively efficient in the number of similarities computed, especially for larger collections.

A comparison of the effectiveness of the different searches is difficult using just the mean values given in Table 4.4. A determination of which search is most efficient depends upon the speed of accessing a page relative to the speed of computing a similarity. This speed ratio is, of course, affected by many other factors: the similarity measure used, the speed of the secondary storage device, the placement of pages on the disk, the use of pre-fetching of pages, *etc.*

A convenient way of working with relative speeds, and thus determining the relative efficiency of the different searches, is to compute the speed ratio that will make two searches take equal amounts of time. That is, if the inverted index search and a cluster search are assumed to take equal amounts of time, then one can find the amount of time it takes to access a page expressed in terms of the number of similarity computations that can be performed in the same amount of time. If fewer than the derived number of similarities can be computed in the time it takes to access a page, the cluster search will take less total time than the inverted file search. Similarly, if more than the derived number of similarities can be computed in the amount of time it takes to access a page, the inverted index search will take less time. Factors that affect the relative speeds of accessing a page and computing a similarity only determine if the necessary speed ratios are physically realizable; they do not affect the ratios themselves.

For example, the mean number of similarities computed in the inverted index search for the INSPEC collection is 4463.4 and the mean number of page accesses is 24.0; for the complete link top-down cluster search the figures are 873.0 and 914.5 respectively. If the two searches take equal time, then

$$\begin{aligned} 4463.4 + 24.0x &= 873.00 + 914.5x \\ 3590.4 &= 890.5x \\ 4.0 &= x\,. \end{aligned}$$

In other words, when a page is accessed in the same amount of time as it takes to compute four similarity computations, the inverted index search and the complete link top-down search take equal amounts of time (for this collection).

Table 4.5 gives the number of similarities that must be computed in the time it takes to access a single page if the total search times of the inverted index and cluster searches are to be equal. The fractional entries in Table 4.5 imply that accessing a page needs to be faster than computing a similarity if the cluster search is to take the same amount of time as the inverted index search. The starred entries correspond to physically unrealizable ratios: accessing a page would need to take zero or negative time.

It is safe to conclude that the cluster searches corresponding to fractional entries are less efficient than the inverted index search since it is highly unlikely that computing a single similarity would be more expensive than accessing a page. However, it is difficult to derive tighter bounds for realistic (obtainable) ratios since the speeds of accessing a page and computing a similarity computation depend on so many factors. Using the implementation of the p-norm similarity measure included in the SMART package, and assuming that thirty page accesses can be performed per second, approximately four similarities were computed in the time it took to access a random page in these experiments. Using the four similarity computations per page access figure,

Table 4.5: Number of time units required to access a page assuming a similarity computation takes one time unit and the cluster search takes the same amount of time as the inverted index search

	(fixed) single link		complete link		group average link	
	bu-indiv	**td-indiv**	**bu-indiv**	**td-indiv**	**bu-indiv**	**td-indiv**
CACM	3.0	0.1	*	0.3	*	1.7
CISI	10.2	0.4	4.4	2.6	3.0	2.8
INSPEC	54.5	5.6	12.1	4.0	2.3	25.2
MED	0.8	0.2	0.5	1.6	*	3.1

* inverted index search clearly more efficient – ratio either approximately zero or negative.

both the single link and the complete link bottom-up searches are at least as efficient as the inverted index search for the CISI and INSPEC collections; for the MED and CACM collections, none of the searches is more efficient than the inverted index search.

The relative efficiency of the cluster searches and the inverted index search varies with the collection. The number of similarities that need to be computed in the time it takes to access a single page (if inverted index and cluster searches are to take equal time) is smaller for the CACM and MED collections than for the INSPEC and CISI collections. The smaller number of similarities required implies the CACM and MED collections have an efficient inverted search while the cluster searches are much more competitive for the CISI and INSPEC collections. As explained earlier, the efficiency of the inverted index search depends on the number and frequency of the query terms. Table 4.6

Table 4.6: Collection characteristics that affect the efficiency of the inverted index search

	mean # terms per query	mean # docs per query term	mean % docs per query term
CACM	4.5	150.2	.05
CISI	7.2	225.9	.16
INSPEC	13.2	950.0	.08
MEDLARS	39.7	28.1	.03

provides a summary of these collection statistics. The CACM queries are both relatively short and occur in relatively few documents. The MED queries are the longest of all the collections, but each query term appears in very few documents on average. The CISI and INSPEC query terms occur in a much larger percentage of the documents.

The single link bottom-up search has the largest ratios of all the hierarchy/search combinations for all collections except MED. The complete link bottom-up search also has fairly large ratios for the INSPEC and CISI collections, while the bottom-up search of the group average link hierarchy has most of the smallest ratios. For all the collections, the bottom-up search of the single link hierarchy is the most efficient of the bottom-up searches, the search of the complete link hierarchy is the next most efficient, and the search of the group average link hierarchy is the least efficient. The difference in the search efficiencies and therefore in the ratios, is the result of the difference in the number of low-level clusters in the hierarchies: the single link hierarchy has the fewest low-level centroids, the complete link hierarchy has the next fewest low-level centroids, and the group average link hierarchy has the most low-level centroids (see Tables 2.4-2.6). The greater the number of low-level clusters, the more expensive the inverted index search of the centroids of those clusters.

The bottom-up search of the MED collection is more inefficient relative to the inverted index search than the same search for the other collections. The inverted index search of the documents of the MED collection is efficient in spite of the large mean query length because each query term appears in few documents on average. A query term occurring in a single document may, however, occur in several centroids since the centroid of an ancestor of a node will contain many of the terms in the descendant centroids. Thus the inverted lists of query terms are likely to stay approximately as long in the inverted index of the low-level centroids as in the inverted index of the documents despite the fact that there are more documents than low-level centroids. The number of similarities computed by each search as given in Table 4.4 shows that for the MED collection the bottom-up search computes nearly as many similarities as the inverted index search. For the CISI and INSPEC collections, the bottom-up search computes many fewer similarities than the inverted index search implying that inverted document lists for the query terms are long enough to make the smaller number of low-level centroids the overriding effect.

The efficiency of the bottom-up searches is unlikely to change very much if centroids of maximum length 100 were used instead of centroids of maximum length 250. Since query terms tend to be frequent terms and the centroids of maximum length 100 would still contain the 100 most frequent terms of the cluster, the query terms are likely to be in almost as many 100-term centroids as in 250-term centroids.

The efficiency of the top-down search of the different hierarchies is precisely the opposite as that for the bottom-up searches: the search of the group average link is the most efficient (has the largest ratios), the search of the complete link hierarchy is the next most efficient (has the next largest ratios), and the search of the single link hierarchy is the least efficient (has the smallest ratios) for all of the collections except INSPEC. For INSPEC, the search of the complete link hierarchy is the most expensive. The difference of the INSPEC collection with respect to the other collections illustrates an important point: the cost of the top-down search of the group average link and single link hierarchies grows sublinearly with collection size. The INSPEC collection is large enough for this effect to become evident.

It was noted in section 3.2.3 that the maximum size of a complete link cluster remains approximately constant, independent of collection size. This fact was used to explain why this search remains effective for the larger collections. A constant depth implies that the number of top-level clusters grows linearly as the collection size increases. This in turn implies that the expense of the top-down search of the complete link hierarchy grows linearly with the collection size since the search begins by computing the similarity between the query and each centroid of the top-level clusters. The cost of the inverted index search also grows linearly as collection size increases – the frequency of query terms grows linearly with collection size – so the relative efficiency of the top-down search of the complete link hierarchy and the inverted index search is unlikely to be affected by collection size. (The expense of the bottom-up search of all the hierarchy types also grows linearly with the collection size since the number of low-level clusters grows linearly.)

The depths of the single link hierarchy and group average link hierarchy increase as the collection size increases. If the top-down search is essentially a depth-first search (as indicated by the experiments), the number of hierarchy nodes expanded in a search, and hence the expense of the search, will grow sublinearly with collection size. For these two hierarchies, then, the top-down search will be more efficient than either of the other searches regardless of the relative speeds of accessing a page and computing a similarity for any sufficiently large collection. Of course, when large collections are used the effectiveness of the top-down search of these hierarchies is smaller than the effectiveness of the other searches. Litofsky and Murray also noted that the relative efficiency of top-down and inverted index searches depends on collection size [Litofsky 69; Murray 72].

4.3.2. Processing Time of Feedback Queries

The results of the previous section demonstrated that the efficiency of the inverted index and bottom-up searches depends on the length of the query while the efficiency of the top-down search does not. If longer queries were to be used, the relative efficiency of the searches may differ. While the initial queries submitted to a retrieval system are usually fairly short, longer queries are formed when *feedback* is used. Since feedback has been shown to be a useful method for improving the search effectiveness [Salton 71a; Salton, Fox & Voorhees 85], the practicality of performing feedback searches should be investigated for the different searching strategies.

Feedback is the process of modifying an existing query using information provided by the user in the form of relevance assessments for previously retrieved documents. In so doing, one hopes to construct queries that more accurately reflect the user's information need. In a vector environment, the query is modified by increasing the weights of terms appearing in most relevant documents, and in addition possibly decreasing the weights of terms that appear in most nonrelevant documents. New terms from the relevant documents may also be added to the query [Salton 71a]. In Boolean (and p-norm) environments, the query is modified by rearranging and adding clauses to the query [Dillon & Desper 80; Salton *et al.* 83; Salton, Fox & Voorhees 85]. The new query is submitted to the system and new documents are retrieved. The feedback process may be repeated several times using the relevance assessments of the newly retrieved documents.

In this section the effectiveness and the efficiency of one iteration of feedback are investigated using the inverted index and individual document searches. The efficiency measures include only the cost of searching the collections using feedback queries; they do not include the cost of creating the queries. The results for both the initial searches and the feedback searches are presented in Tables 4.7-4.10. The same evaluation measures are used for each iteration: the effectiveness is measured by the precision after ten and twenty documents are retrieved ($NumWanted=20$ for both iterations), the total number of relevant documents retrieved, and the total number of queries retrieving no relevant documents; the efficiency is measured by the number of pages accessed and the number of similarities computed. Also included in the Tables is the percentage change in the precision after ten documents are

Table 4.7: CACM feedback effectiveness and efficiency comparisons

access method	search strategy	original queries prec	original queries tot. rel.	original queries # q's no rel	original queries % change	feedback queries prec	feedback queries tot. rel.	feedback queries # q's no rel	feedback queries % change
inverted index		.256	128	11	–	.378	189	5	–
		.225	225	5		.273	273	3	
single link	bu-indiv	.190	95	18	−25.8	.290	145	11	−23.3
		.160	160	15		.196	196	10	
	td-indiv	.150	75	18	−41.4	.250	125	13	−33.9
		.092	92	16		.154	154	10	
complete link	bu-indiv	.260	130	11	+1.6	.364	182	7	−3.7
		.207	207	10		.257	257	6	
	td-indiv	.286	143	10	+11.7	.374	187	2	−1.0
		.232	232	4		.274	274	1	
group ave link	bu-indiv	.264	132	10	+3.1	.378	189	8	0.0
		.223	223	9		.269	269	7	
	td-indiv	.264	132	13	+3.1	.380	190	7	+0.5
		.224	224	9		.270	270	3	

a) effectiveness statistics

access method	search strategy	original queries mean sims	original queries mean pages	original queries ratio of speeds	feedback queries mean sims	feedback queries mean pages	feedback queries ratio of speeds
inverted index		563.0	8.7	–	652.6	11.2	–
single link	bu-indiv	346.0	81.3	3.0	389.4	115.0	2.5
	td-indiv	505.4	528.8	0.1	592.7	622.0	0.1
complete link	bu-indiv	561.6	68.3	*	658.8	86.3	*
	td-indiv	408.9	452.5	0.3	430.9	490.0	0.5
group ave link	bu-indiv	649.1	73.9	*	774.4	97.1	*
	td-indiv	168.9	247.6	1.7	203.2	303.0	1.5

b) efficiency statistics

Table 4.8: CISI feedback effectiveness and efficiency comparisons

access method	search strategy	original queries prec	tot. rel.	# q's no rel	% change	feedback queries prec	tot. rel.	# q's no rel	% change
inverted index		.254	89	9	-	.391	137	6	-
		.244	171	5		.293	205	4	
single link	bu-indiv	.260	91	13	+2.4	.346	121	9	+1.3
		.199	139	11		.241	169	7	
	td-indiv	.229	80	10	−9.8	.343	120	6	−12.3
		.191	134	8		.240	168	5	
complete link	bu-indiv	.271	95	11	+6.7	.403	141	6	+3.1
		.210	147	8		.289	202	6	
	td-indiv	.286	100	11	+12.6	.429	150	5	+9.7
		.256	179	4		.311	218	3	
group ave link	bu-indiv	.280	98	11	+10.2	.391	137	9	0.0
		.211	148	9		.281	197	7	
	td-indiv	.271	95	9	+6.7	.434	152	6	+11.0
		.243	170	6		.306	214	4	

a) effectiveness statistics

access method	search strategy	original queries mean sims	mean pages	ratio of speeds	feedback queries mean sims	mean pages	ratio of speeds
inverted index		913.9	13.7	-	930.0	14.9	-
single link	bu-indiv	460.6	58.1	10.2	476.3	81.8	6.8
	td-indiv	663.6	692.4	0.4	678.7	712.2	0.4
complete link	bu-indiv	716.3	58.5	4.4	730.6	70.2	3.6
	td-indiv	224.5	281.2	2.6	256.1	333.0	2.1
group ave link	bu-indiv	777.7	59.7	3.0	792.1	72.2	2.4
	td-indiv	183.9	271.7	2.8	210.9	316.4	2.4

b) efficiency statistics

Table 4.9: INSPEC feedback effectiveness and efficiency comparisons

access method	search strategy	original queries prec	original queries tot. rel.	original queries # q's no rel	original queries % change	feedback queries prec	feedback queries tot. rel.	feedback queries # q's no rel	feedback queries % change
inverted index		.348	268	13	-	.453	349	4	-
		.294	452	8		.320	492	4	
single link	bu-indiv	.279	215	15	−19.8	.364	280	12	-19.6
		.205	315	13		.233	358	9	
	td-indiv	.056	43	51	−83.9	.096	74	42	−78.8
		.036	56	46		.059	91	40	
complete link	bu-indiv	.343	264	12	−1.4	.438	337	7	−3.3
		.255	393	8		.303	467	6	
	td-indiv	.364	280	11	+4.6	.466	359	6	+2.9
		.288	443	6		.323	498	4	
group ave link	bu-indiv	.339	261	11	−2.6	.435	335	8	−4.0
		.253	389	8		.301	464	7	
	td-indiv	.253	195	29	−27.3	.360	277	17	−3.7
		.192	295	22		.248	382	16	

a) effectiveness statistics

access method	search strategy	original queries mean sims	original queries mean pages	original queries ratio of speeds	feedback queries mean sims	feedback queries mean pages	feedback queries ratio of speeds
inverted index		4463.4	24.0	-	4833.8	26.8	-
single link	bu-indiv	1967.3	69.8	54.5	2088.0	84.4	47.7
	td-indiv	684.2	700.4	5.6	902.8	921.6	4.4
complete link	bu-indiv	3871.0	73.1	12.1	4226.4	87.7	10.0
	td-indiv	873.0	914.5	4.0	886.3	938.3	4.3
group ave link	bu-indiv	4338.4	78.6	2.3	4701.6	92.1	2.0
	td-indiv	133.6	195.6	25.2	143.9	214.6	25.0

b) efficiency statistics

Table 4.10: MED feedback effectiveness and efficiency comparisons

access method	search strategy	original queries prec	tot. rel.	# q's no rel	% change	feedback queries prec	tot. rel.	# q's no rel	% change
inverted index		.577	173	1	-	.817	245	0	-
		.518	311	0		.620	372	0	
single link	bu-indiv	.597	179	2	+3.5	.780	234	2	−4.5
		.490	294	2		.572	343	2	
	td-indiv	.417	125	5	−27.7	.620	186	3	−24.1
		.362	217	2		.503	302	3	
complete link	bu-indiv	.593	178	2	+2.8	.790	237	2	−3.3
		.533	320	2		.633	380	2	
	td-indiv	.600	180	1	+4.0	.823	247	1	+0.7
		.553	332	0		.653	392	1	
group ave link	bu-indiv	.607	182	1	+5.2	.793	238	1	−2.9
		.552	331	1		.637	382	1	
	td-indiv	.590	177	3	+2.3	.787	236	2	−3.7
		.540	324	3		.653	392	2	

a) effectiveness statistics

access method	search strategy	original queries mean sims	mean pages	ratio of speeds	feedback queries mean sims	mean pages	ratio of speeds
inverted index		367.7	36.2	-	443.9	40.6	-
single link	bu-indiv	312.3	108.1	0.8	360.9	116.1	1.1
	td-indiv	308.2	330.2	0.2	337.5	363.9	0.3
complete link	bu-indiv	345.7	83.0	0.5	422.6	156.1	0.2
	td-indiv	144.5	179.8	1.6	94.8	201.7	2.2
group ave link	bu-indiv	367.6	84.4	*	445.0	96.5	*
	td-indiv	85.6	128.4	3.1	96.4	148.7	3.2

b) efficiency statistics

retrieved between the inverted index and the given cluster searches, and the amount of time a page access must take if the inverted index search and the given cluster search are to take the same amount of time. These figures are always computed using searches of the same iteration.

For each searching strategy, the feedback queries were created using the first ten documents returned by that search. These feedback queries were then processed using the same search strategy used in the initial search. Since in general the different searching strategies return different documents, the queries used in the feedback iteration for the different searches may differ. The algorithm used to create the p-norm feedback queries was developed by Fox [Fox 83a; Salton, Fox & Voorhees 85]. The new query is constructed as a disjunction of single terms, pairs of terms, and triples of terms, with each clause selected on the basis of its weight and expected frequency. The query is designed to retrieve fifty documents for the smaller collections and one hundred documents for the INSPEC collection. Terms that occur in the original query are weighted as if they occurred in two relevant documents. The final query is the disjunction of the newly created query and the original query.

The effectiveness of the searches is evaluated using *partial rank freezing*. This method is a variant of the modified freezing method described by Chang, Cirrillo, and Razon [Chang, Cirrillo & Razon 71]. Partial freezing modifies the ranking of the documents returned by earlier iterations of the search; once the ranking is modified, the evaluation measures are computed in the usual way. The ranking of the documents of the current iteration is produced as follows:

- the relevant documents that were returned to the user in previous iterations keep the same ranks as the iteration in which they were first returned;
- the nonrelevant documents that were returned in the previous iterations are placed in the largest ranks;
- the documents that are returned in the current iteration "fill in the holes" around the frozen relevant documents.

Figure 4.1 contains an example of the ranking produced by one iteration of feedback using partial rank freezing.

Since the relevant documents of earlier iterations are frozen into their ranks, the values of the evaluation measures for a ranking produced by partial freezing must be at least as good as the values for the previous iteration. In this way, partial freezing models the effectiveness of the search as seen by the user accurately. More importantly for these experiments, evaluating the effectiveness after partial freezing guarantees that the same number of queries are used in the evaluation of the different searches.

Documents that are used to create the feedback query (the first ten documents) are only used in the evaluation of subsequent iterations; they are returned by the search procedure exactly once. If in the course of an inverted index search the id of a document that was used in a previous iteration is encountered in an inverted list, the next document id is processed immediately. The top-down search never adds the similarity of a document that was used to create the feedback query in a previous iteration to the heap, thereby preventing that document from being retrieved again. The bottom-up

rank	initial ranking	ranking returned by feedback search	final partial freezing ranking
1	*A*	*F*	*F*
2	**B**	**J**	**B**
3	*C*	*H*	**J**
4	**D**	*I*	**D**
5	*E*	*G*	*H*
6	*F*		*I*
7	*G*		*G*
8	*H*		*A*
9	*I*		*C*
10	**J**		*E*

10 documents in collection, 5 documents returned per iteration;
bold docs are relevant; *italicized* docs are nonrelevant

Figure 4.1: Ranking produced by partial rank freezing

search retrieves the top forty centroids and expands the clusters corresponding to the top ten centroids by computing the query-vector similarity of all documents in the cluster that were not previously used to create the feedback query. If the number of documents available to be returned after the first ten clusters are expanded is less than the number wanted, the next clusters are expanded one at a time until either the desired number of documents is seen or all forty clusters are expanded.

The efficiency statistics in Tables 4.7-4.10 show that the feedback search is more expensive than the initial search for all searching strategies. Both the number of similarities computed and the number of pages accessed are greater in the feedback iteration than in the initial iteration for all searches. The relative efficiency of the inverted index and cluster searches is approximately the same for both iterations. In most of the cases in which there is a difference, the initial cluster searches are less expensive when compared to the initial inverted index searches than are the feedback cluster searches when compared to the feedback inverted index searches (recall that a smaller speed ratio implies the inverted index search is more efficient).

The increase in the expense of the feedback inverted index and bottom-up searches was expected – the expense of those searches is proportional to the length of the queries. The longer feedback queries caused more similarities to be computed and more pages to be accessed as expected.

The expense of the top-down search is not dependent on the query length. However, the expense of that search is dependent on the number of documents to be retrieved; when more documents are wanted, more hierarchy nodes must be expanded which results in more page accesses and more similarity computations. Each feedback search is in effect asking for a larger number of documents to be retrieved despite a constant value for *NumWanted* because documents returned in previous iterations must be ignored. Since in general the clusters that contain previously retrieved documents are the very tiny low-level clusters that seldom contain other non-retrieved documents, the feedback top-down search must expand more nodes than the initial top-down search to retrieve *NumWanted* new documents. Thus the feedback search is more expensive than the initial search

The difference between the effectiveness of the cluster feedback searches and the inverted index feedback search is usually smaller than the difference for the initial searches. This is as expected: all the search strategies used rank individual documents by their similarity to the query; as more documents are examined, the rankings produced by the different strategies must converge. Note, however, that many of the cluster feedback searches are less effective, or only slightly more effective, than the inverted index feedback searches.

In these experiments, the relative efficiency of inverted index and cluster searches remained approximately constant in the feedback iteration while the relative effectiveness of the inverted index search improved after the feedback iteration. These results suggest that the inverted index search may be as effective as each of the cluster searches after several iterations of feedback.

4.3.3. Comparisons Using Packed Vectors

The processing time experiments discussed so far have investigated the efficiency of all the hierarchies and both bottom-up and top-down searching strategies. The top-down search of the single link and group average link hierarchies is the most efficient search asymptotically, and the bottom-up search of the single link and complete link hierarchies is generally efficient compared to the inverted index search if the query terms are fairly frequent. The only search that was consistently at least as effective as the inverted index search, however, was the top-down search of the complete link hierarchy. Therefore, the remainder of the processing time experiments will concentrate on the efficiency of the inverted index search and the top-down search of the complete link hierarchy. In what follows, "the cluster search" refers to this search.

Both the effectiveness and the relative efficiency of the top-down search of the complete link hierarchy has been explained by the very large number of top-level clusters. The search spends most of its time accessing the pages that contain the top-level centroids. For example, at least 802 (the number of top-level clusters) of the 914.5 pages that are accessed on average in the INSPEC search are accessed to process the top-level centroids.

Much of the space inside a page is wasted, however; since centroids are limited to 250 terms and each term and its weight use a total of eight bytes, approximately half of a 4096 byte page is unused. Also, given the way the top-down search is defined, putting more centroid and document vectors onto the same page must reduce the mean number of pages accessed in a search. The top-down search never accesses a few children of a node, it either accesses none of the children or it accesses all of the children. Therefore, packing the vectors associated with hierarchy nodes that are siblings into as few pages as possible should result in a substantial decrease in the mean number of pages accessed in a search as compared to the current configuration. (This new packed configuration of the hierarchy can still be updated efficiently since only vectors associated with siblings are packed into the same pages.)

Table 4.11 contains the mean number of pages accessed in the cluster search before and after vectors are packed into pages and in the inverted index search. No change was made to the configuration of the inverted index pages since the inverted index is accessed randomly. As expected, the mean number of pages accessed by the cluster search for packed vectors is significantly less than the mean number accessed when each vector is started on a new page.

Table 4.11: Efficiency of top-down search with packed vectors

	mean number of pages accessed			ratio of speeds	
	inverted index	cluster (not packed)	cluster (packed)	not packed	packed
CACM	8.7	452.5	158.3	0.3	1.0
CISI	13.7	281.2	162.4	2.6	4.6
INSPEC	24.0	914.5	396.0	4.0	9.7
MED	36.2	179.8	104.5	1.6	3.3

The decrease in the number of pages accessed is reflected in the larger ratios of the speeds of accessing a page to computing a similarity needed to make the inverted index and cluster searches take equal amounts of time. These ratios are given on the right side of Table 4.11. Assuming four similarities can be computed in the time it takes to access a single random page, the cluster search now uses less time on average than the inverted index search for the INSPEC collection. The cluster and inverted index searches require approximately the same amount of time for the CISI and MED collections. The inverted index search is more efficient for the CACM collection.

The experiments suggest that the top-down search of the complete link hierarchy that retrieves individual documents from within clusters is more effective than the inverted index search and, for collections that have long inverted lists, is also more efficient than the inverted index search. These experiments, however, were performed using an expensive similarity measure.

As an example, consider an environment in which a machine instruction takes one microsecond to execute and a page access takes 30,000 machine instructions (so approximately 30 pages can be accessed per second). This environment is similar to the environment in which these experiments took place. If four similarities can be processed in the time it takes to access a page, an average similarity computation requires 7,500 instructions. Table 4.12 contains the mean number of seconds needed to process a query in this environment. The packed representation of the clustered hierarchy is used in obtaining these figures. For example, the time to process a query for the clustered search of the INSPEC collection is calculated as

$$(396.0\text{pages} \times .03\text{sec/page}) + (873.0\text{sims} \times 7500\text{instr/sim} \times .000001\text{sec/instr}).$$

The mean number of seconds to process a query as given in Table 4.12 illustrates the difference in the efficiency of the clustered and inverted index searches. Indeed, the mean times of the two searches are similar for the CISI and MED collections, the inverted index search is faster for the CACM collection, and the clustered search is faster for the INSPEC collection. The times also illustrate the increase in search time as the collection size increases – the time needed to search the larger INSPEC collection is significantly longer than the time needed to search the other smaller collections. Since both searches will take correspondingly longer on collections larger than INSPEC, it is worthwhile to investigate the performance of the searches using a less expensive similarity measure. This topic is explored in the next subsection.

4.3.4. Comparisons Using an Inexpensive Similarity Measure

Most of the processing time of the inverted index search is spent in performing similarity computations. For the p-norm similarity measure, the similarity between the query and a vector that has only one term in common

Table 4.12: Mean number of seconds to process a query using the p-norm similarity measure assuming thirty pages accessed per second, four similarities computed per page access, and one microsecond per instruction

	inverted index search	clustered search
CACM	4.5	7.8
CISI	7.3	6.6
INSPEC	34.2	18.4
MED	3.8	4.2

with the query is just as expensive to compute as the similarity between the query and a vector that contains all the query terms. A single very frequent query term will make the search for that query very expensive. However, for other similarity measures, namely those that can be computed using the inner product between two vectors, the inverted index search can be made much more efficient.

In the inner product version of the inverted index search, the similarity between the query and each of the documents is accumulated in term order instead of document order. That is, for each document a memory location is used to store the current similarity of that document with the query. (Note that this assumes that $4N$ bytes of memory are available where N is the number of documents in the collection.) Each of the inverted lists for the query terms is traversed, and the weight of the term in the current document of the current list is added to the current similarity for that document (see Figure 4.2). With this inverted index algorithm, only a few machine instructions are performed for each document id in the inverted lists (including duplicates),

```
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)
}
```

Figure 4.2: Idealized machine instructions of inverted index search

instead of the thousands of machine instructions required to perform each complete p-norm similarity computation.

The number of query-vector similarities performed cannot be used to measure the CPU time required by a search for this algorithm. Instead, the number of idealized machine instructions performed is used as the measure. Each (floating point) addition, (floating point) multiplication, comparison, pointer increment, and array indexing is counted as one operation. The processing time experiments are repeated for the top-down complete link search and the inverted index search, using the algorithms (annotated with instruction counts) shown in Figures 4.2 and 4.3. The inner product similarity measure is used with both searches; the weights of the terms in the vectors are normalized, however, so that the similarity computed is equal to the cosine similarity measure. (Since the inner product similarity measure is used, vector queries, not p-norm queries, are used in these experiments. The vector queries were produced from the natural language statements submitted by the users from which the p-norm queries used earlier were constructed.) 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 4.2,

```
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;
        }
    }
}
```

Figure 4.3: Idealized machine instructions of top-down cluster search

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, the main part is computing the similarity between a given query and document or centroid vector (Figure 4.3). The cluster search performs three instructions for each term that appears in exactly one of the query and

Table 4.13: Processing time experiments using inner product similarities

	inverted index search					clustered search				
	mean prec	# rel ret	# q's no rel	mean # instr.	mean pages	mean prec	# rel ret	# q's no rel	mean # instr.	mean pages
CACM	.230 .185	115 185	9 5	9,296	20.3	.248 .199	124 199	8 6	129,230	155.1
CISI	.269 .260	94 182	7 5	8,339	15.1	.274 .237	96 166	6 3	105,583	158.2
INSPEC	.384 .315	296 485	3 2	67,877	48.9	.373 .284	287 437	6 4	517,999	418.3
MED	.620 .528	186 317	0 0	3,136	18.1	.650 .603	195 362	0 0	71,632	106.5

centroid/document vectors and six instructions when a match occurs. Note that the cluster search has no way of avoiding computing similarities of zero.

The results of the cosine experiments are included in Table 4.13. The effectiveness measures used are the mean precision, the total number of relevant documents retrieved, and the total number of queries retrieving no relevant documents after 10 and 20 documents are retrieved. The efficiency measures used are the mean number of idealized machine instructions performed per query and the mean number of pages accessed per query.

The cosine results differ from the p-norm results in two important ways. First, the inverted index search is more efficient than the inverted index search: both machine instructions and pages accesses are (much) greater for the cluster search than for the inverted index search. Second, for the INSPEC collection the inverted index search is more effective than the cluster search.

Although some difference in the efficiency of the searches was expected, the actual difference in the efficiency of the searches is dramatic: 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 this inverted index implementation takes full advantage of the sparsity of the document vectors. Not only are zero similarities not computed, but terms that do not appear in both the document (or centroid) and the query are also ignored. Although the cluster search avoids the computation of the similarity of the query with a large percentage of the documents, the remaining documents and centroids must be exhaustively examined.

Using the same assumptions as before, namely that 30 pages can be accessed per second and that a machine instruction takes 1 microsecond, the mean number of seconds per search can be computed. Table 4.14 contains these mean times for the inner product experiments.

It is difficult to make direct comparisons between the figures given in Tables 4.12 and 4.14 since several factors – the query type, the similarity measure, and the implementation of the inverted index search – are different in the different environments. One can conclude that the inner product inverted index search is the most time efficient of the searches investigated in this thesis that are of comparable effectiveness.

The difference in the effectiveness of the inverted index and clustered searches when the inner product similarity measure is used is harder to explain. Further testing on large collections is necessary to determine if the

Table 4.14: Mean number of seconds to process a query using the inner product similarity measure assuming thirty pages accessed per second, four similarities computed per page access, and one microsecond per instruction

	inverted index search	clustered search
CACM	.62	4.78
CISI	.46	4.85
INSPEC	1.54	13.07
MED	.55	3.27

difference is related to the size of INSPEC or some other characteristic of the collection. This illustrates an important disadvantage of cluster searching: any seemingly minor change (in this case a change in the similarity measure) may have unexpected repercussions.

4.4. Conclusion

The efficiency of inverted index and cluster searches was investigated in this chapter. The efficiency was measured by both the amount of disk space required to store the auxiliary data structures and the mean time required to process a query. The processing time was measured as the sum of the CPU time, measured by the number of similarity computations performed, and the I/O time, measured by the number of disk pages accessed.

For each collection tested, the complete link and group average link hierarchies (including the centroids) required more disk space than the inverted index of the document vectors. The bulk of the space was used to store the centroids. The single link hierarchy was usually smaller than the inverted index since the fixed single link hierarchy (which represents fewer clusters in the hierarchy) was used.

Which of the inverted index or cluster searches has the shortest mean processing time depends on many factors. For sufficiently large collections, the top-down search of the single link and group average link hierarchies have shorter mean processing times than the other searches regardless of any other factor since the running time of the top-down search on those two hierarchies increases sublinearly with collection size while the running time of the other searches increases linearly with collection size. How large is sufficiently large depends upon the frequency of query terms and the relative cost of accessing a page and computing a similarity.

For the collections used in the experiments, the relative efficiency of the searches depends on the ratio of the time required to access a page to the time required to perform a similarity computation. Larger ratios favor the cluster searches since those searches access more pages but compute fewer similarities than the inverted index search. The relative efficiency of the searches remained the same when p-norm feedback queries were used.

The inverted index search was relatively efficient, especially for collections in which query terms appear in few documents on average or when efficient similarity measures were used. The bottom-up search of the (fixed) single link hierarchy and, to a lesser degree, the complete link hierarchy was more time efficient than the inverted index search and the other cluster searches for the INSPEC and CISI collections. The query terms of these collections appear in relatively many documents, so the query term lists in the inverted index of low-level centroids are shorter than the lists in the document inverted index. The group average hierarchy contains many low-level clusters; hence the bottom-up search of that hierarchy was the most expensive cluster search.

The efficiency of the top-down search of the complete link hierarchy is of particular interest since that search was the most effective search in the experiments of Chapter 3. Although a fairly small percentage of the total number of documents in the collection were accessed in this search, the inverted index search both used many fewer machine instructions and accessed many fewer pages when the inner product similarity measure was used. When the more expensive p-norm similarity measure was used, the cluster search was more efficient than the inverted index search.

These conclusions contradict the conclusions drawn by Murray when he contrasted inverted index and cluster searches [Murray 72]. He concluded:

1) the inverted index search achieved higher precision;

2) the inverted file search required more disk space than the cluster search when the direct document file was included in the space requirements ;

3) the time required by the inverted index search depended on the query length and number of documents to be retrieved, while the time required by the cluster search depended only on the number of nodes to be expanded, not on the number of documents to be retrieved; and

4) the inverted file search required more disk accesses .

The conflict between these conclusions and the conclusions reached in this thesis is the result of different assumptions and, more importantly, the difference in the hierarchies used.

As was explained in section 1.1.2, the hierarchy created by the heuristic clustering algorithm used by Murray and the agglomerative clustering methods are very different. The hierarchy used by Murray contains relatively few, large clusters while the agglomerative hierarchies contain many clusters of widely varying sizes. In particular, the complete link and group average link hierarchies contain many very small clusters.

The difference in the size of the clusters used explains why Murray found his inverted index search to produce higher precision than his cluster search while the opposite was found for the complete link hierarchy in this work. The results of previous work by the SMART group [Grauer & Messier 71] and of the experiments comparing the single link hierarchy to the complete link and group average link hierarchies in Chapter 3 demonstrated that smaller clusters produce higher precision searches than larger clusters.

The difference in the number of clusters contained in the hierarchies explains why Murray found the inverted index to take more disk space than the cluster hierarchy and associated centroids while the inverted index took less space than the agglomerative hierarchies and their centroids. Few clusters implies few centroids and thus a small direct centroid file.

The remaining conflicting conclusions are explained not only by the size of the clusters, but also by a different assumption. Murray assumed that the result of a search was a list of document citations. Since the inverted index did not contain these citations (and therefore needed a disk accesses to fetch each one) while the cluster file did contain the citations (and required only one disk access to fetch many citations), the cost of presenting the output was included in the cost of the search. I assumed the output of a search was a natural language title (and abstract) of the retrieved documents and that the cost of presenting the same number of documents was approximately equal for the two searches. As described earlier, this assumption is slightly 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.

Murray's top-down search required fewer disk accesses than the inverted index search only when the cost of presenting the citations was included. Furthermore, the presentation of the results was the only component of the inverted index search that depended on the number of documents desired. The top-down search did not depend on the number of documents desired because the clusters were so large. A (small) change in the number of documents desired did not cause any more hierarchy nodes to be expanded.

The top-down search of the agglomerative hierarchies, however, required many disk accesses due to the number of clusters in the hierarchy. Since the low-level clusters are small, each increment in the number of documents desired causes more hierarchy nodes to be expanded, causing the running time of the top-down search to depend on the number of documents desired. The inverted index search ranks the entire collection regardless of the number of documents wanted, so a larger number of documents can be retrieved (essentially) for free.

Chapter 5
Conclusion

The first goal of this thesis was to systematically compare different clustering methods and clustered search strategies. The results of these comparisons can be summarized by considering the variables that were examined.

Three different hierarchy types, the single link, the complete link, and the group average link hierarchies, were tested. The single link hierarchy is different from the other two hierarchies in that the low-level clusters are larger on average in the single link hierarchy. As a result, the searches of the single link hierarchy are less effective than searches of the other hierarchies. The single link hierarchy contains fewer clusters than the other hierarchies, however, so it is the most space efficient hierarchy. The group average link hierarchy contains the most clusters, and thus it is the least space efficient hierarchy.

Differences among the hierarchy types were shown to depend on the size of the collection. The group average link hierarchy is similar to the complete link hierarchy for small collections, but differences among the three hierarchy types increase as the collection size increases. Most of the differences seem to be caused by the difference in the depths of the hierarchy as the collection size increases. For the single link and group average link hierarchies, the depth increases as the collection size increases, while the depth of the complete link hierarchy is bounded in practice, regardless of collection size.

Both bottom-up and top-down searching strategies were used in the experiments. The bottom-up search was more effective than the top-down search except for the complete link hierarchy. This conclusion is in accord with the results of Croft who found bottom-up searching of the single link hierarchy was expected to be more effective than top-down searches due to the uncertainty involved at high levels of the hierarchy [Croft 80]. Since the group average link and the single link hierarchies grow deeper as the collection grows larger, the uncertainty increases as the collection gets larger and the top-down search of these hierarchies becomes less effective. The top-down search of these hierarchies is the most efficient search asymptotically for the same reason. The increase in the depth of the hierarchy as the collection size increases implies that the processing time of the top-down search grows sublinearly with collection size; for all other searches, including the top-down search of the complete link hierarchy, the search time grows linearly with collection size. The restricted depth of the complete link hierarchy, coupled with the large number of top-level clusters that are a result of the restricted depth, makes the top-down search of the complete link hierarchy effective. Since the depth is independent of collection size for this hierarchy, the top-down search of the complete link hierarchy is likely to remain effective as the collection size increases.

Two selection strategies were tested: retrieving entire clusters and retrieving individual documents from within clusters. Retrieving entire clusters was (much) less effective and more efficient than retrieving individual documents from within clusters. The effectiveness results suggest that the cluster hypothesis does not characterize collections to a sufficient extent to make cluster based retrieval worthwhile.

In general, the clustered search experiments demonstrated that the single link hierarchy is not the hierarchy of choice for information retrieval. Searches of the group average link and complete link hierarchies perform similarly for small collections; for larger collections, searches of the complete link hierarchy are more effective than the searches of the group average link hierarchy, while top-down searches of the group average link hierarchy are more efficient than searches of the complete link hierarchy.

The second goal of the thesis was to compare the efficiency and effectiveness of the inverted index search to the performance of the best clustered searches in order to determine under what circumstances one search is to be preferred over the other. The top-down search of the complete link hierarchy that retrieves individual documents from within clusters is both more effective and more efficient than the inverted index search for these collections when the p-norm similarity is used. However, when the more efficient inner product similarity measure is used with vector queries, the inverted index search is to be preferred for the following reasons:

- The inverted index search is more efficient (it requires less disk space and has a smaller mean processing time per query) than the cluster searches for all but extremely large collections. The top-down search of the group average link or single link hierarchies requires less time for the very large collections, but the top-down search is very ineffective on hierarchies of that size and the hierarchies themselves are difficult to construct for massive collections.

- Although the top-down search that retrieves individual documents from within complete link clusters is more effective than the inverted index search for the collections, the difference in the effectiveness decreases after one iteration of feedback. This suggests that several iterations of inverted index feedback would be as effective as any of the cluster searches used in these experiments. Also, the top-down search was more effective only for high precision searches. It should be noted, however, that fewer queries retrieved no relevant documents in the top-down search than in the inverted index search for the larger collections.

- The inverted index search is easy to implement and depends on fewer parameters than the cluster searches. The effectiveness of the inverted index search depends only on the number of documents that are returned to the user. The effectiveness and efficiency of the cluster searches depend on such factors as the length of the centroids, the weights used within the centroids, and the precision to which the inter-document similarities are computed, in addition to the parameters discussed above. Good values for these parameters must usually be determined experimentally. Poor parameter values may result in very inefficient or ineffective searches. The top-down search of the single link hierarchy (on medium-sized collections) is an example of a search that is both ineffective and expensive compared to the other cluster searches or the inverted index search.

- Creating the cluster hierarchies is expensive, requiring O(N^2) time and, for the complete link hierarchy, O(N^2) space in the worst case. The sorted inverted index assumed in these experiments can be created in $O(N \log N)$ time and $O(N)$ space in the worst case.

The main disadvantage of an inverted index search is that the relationships between documents are completely ignored. Thus an inverted file implementation has no facility for allowing users to browse through a document collection, that is, to find documents that are similar to a given document.

An area for further study is defining retrieval strategies that incorporate (some of) the relationships between documents while retaining the efficiency and robustness of the inverted index search. Griffiths, Luckhurst, and Willett have recently experimented with one such method that is based on the observation that it is the small low-level clusters of the complete link and group average link hierarchies that cause searches of those hierarchies to be effective [Griffiths, Luckhurst & Willett 85]. Their method consists of performing an inverted index search of the nearest neighbor clusters of a collection (clusters that consist of a document and its nearest neighbor). In preliminary testing, the effectiveness of this method was approximately the same as the inverted index search of the document vectors. This method does not facilitate browsing either, since finding only one other document similar to a given document is usually not sufficient.

Another method by which document relationships could be incorporated into a retrieval strategy is to use the complete link hierarchy in a feedback search. In this search, the relevant documents returned by an initial inverted index search would be located in the hierarchy, and documents that were closely related to the relevant documents would be returned. This method has the advantage that the hierarchy is not searched, so centroids are not needed. Furthermore, since different searching strategies usually retrieve different relevant documents even when the total number of relevant documents retrieved is approximately equal [Katzer, *et al.* 82], using two different strategies for the initial and feedback searches should be more effective than using either strategy alone. (Griffiths, Luckhurst and Willett tried combining the results of the inverted index search of the nearest neighbor clusters and the inverted index search of the documents in the initial search. The results showed little difference in total effectiveness of the combined strategy over the effectiveness of either search strategy alone, but fewer queries retrieved no relevant documents in the combined search.) Since the cluster hierarchy is preserved, browsing is possible in this system. The disadvantage of this method is the extra space required to store the hierarchy.

Clustering documents based on the similarity of their content is not the only mechanism that can be used to incorporate document relationships into a searching strategy. Documents have been clustered on the basis of other measures such as bibliographic coupling (two documents are linked if a third document cites both of them) [Kessler 62] or co-citation strength (two documents are linked if both cite a third document) [Small 73]. Alternatively, Kwok has suggested including the terms that occur in the titles of documents that cite a source document in the vector of the source document, thereby eliminating the clustering step [Kwok 84]. The results of experiments using vectors extended by citation data and suggestions for further work in this area are given by Fox [Fox 83a].

The theory of centroid definition and weighting is another area in which further study is needed. If centroids could be defined such that a cluster search could discriminate among large clusters, the top-down search of the group average link hierarchy would be an efficient and effective search.

Appendix
Clustering and Searching Algorithms

This appendix contains a more complete description of the procedures to construct and search the clustered collections described in Chapter 2. The description in Chapter 2 gives the main idea and an example of each algorithm. That description should be understood before this appendix is read. Figure A.1 contains a list of the notation used in the Appendix.

A.1. Clustering Methods

The implementations of the single link, the group average link, and the complete link clustering methods are described in this section. No particular representation of the cluster hierarchy is assumed in this discussion. It is assumed that a procedure for inserting a document into a hierarchy at a given similarity level and a procedure for merging two clusters at a given similarity level while retaining the structure of the clusters being merged are available.

A.1.1. The Single Link Implementation

The single link construction algorithm is an implementation of Prim's algorithm for constructing maximum spanning trees [Tarjan 83]. The hierarchy is created by inserting the document with maximum similarity with a document already in the hierarchy into the hierarchy until all the documents are in the hierarchy. The code for this algorithm can be found in Figure A.2.

The single link construction procedure makes use of two other procedures, ComputeSims(did) and InsertHierarchy(did1,did2,sim). ComputeSims calculates the similarity between document *did* and all the other documents in the

x	variable x
x[y]	array x with index y
x.y	field y of structure (record) x
x(y)	call to procedure x with argument y
=	test for equality
←	assignment
{...}	compound statement
/*...*/	comment
FALSE, TRUE	logical constants
UNDEF	constant denoting an undefined value

Figure A.1: Notation used in algorithms

collection, putting the similarity between document *did* and document *j* in the *j*th element of global array *sims*. In the current implementation, this procedure uses an inverted index of the document collection to avoid computing non-zero similarities. InsertHierarchy makes the current state of the hierarchy consistent with the fact that documents *did1* and *did2* have similarity *sim*.

A.1.2. The Group Average Link Implementation

The implementation of the group average link hierarchy construction algorithm assumes that the similarity between documents can be computed using the inner product similarity function. For the inner product similarity measure, the mean similarity between a document and the documents in a cluster is equal to the similarity between the document and the centroid of the cluster. Thus if the inner product similarity measure is used, the similarity between two clusters can be computed using the centroids of the clusters.

```
for (i ← 2 to CollectionSize) {
    info[i].sim ← 0.0;
    info[i].InHierarchy ← FALSE;
    info[i].nn ← UNDEF;
}
initialize hierarchy by placing document 1 in the hierarchy;
CurrentDid ← 1;

/* place the document having maximum similarity with a document in the
 *      hierarchy into the hierarchy until all documents are in the hierarchy
*/
    while (CurrentDid ≠ UNDEF) {
        info[CurrentDid].InHierarchy ← TRUE;
        ComputeSims(CurrentDid);
        MaxSim ← 0.0;   NextDid ← UNDEF;
        /* update nearest neighbor of each document not yet in hierarchy */
            for (i ← 1 to CollectionSize) {
                if (not info[i].InHierarchy) {
                    if (sims[i] > info[i].sim) {
                        info[i].sim ← sims[i];
                        info[i].nn ← CurrentDid;
                    }
                    if (info[i].sim > MaxSim) {
                        MaxSim ← info[i].sim;
                        NextDid ← i;
                    }
                }
            }
        if (NextDid ≠ UNDEF) {
            InsertHierarchy(NextDid,info[NextDid].nn,MaxSim);
        }
        CurrentDid ← NextDid;
    }
```

Figure A.2: Constructing the single link hierarchy

The group average link procedure is given in Figure A.3. The main loop of the program merges the two clusters that have the maximum similarity until only one cluster remains. This is accomplished using two other procedures, FindMaxSim(cid,nn,sim) and MergeCentroids(cid1,cid2). Initialization is accomplished through the use of a third procedure ComputeSim(did,nn,sim).

ComputeSim calculates the similarity between document *did* and the other documents in the collection. It returns the id of the document with maximum similarity with *did* in *nn* (nearest neighbor) and the similarity between the two documents in *sim*. The current implementation of this procedure uses an inverted index of the document collection to avoid computing zero similarities.

FindMaxSim is similar to ComputeSim. It computes the similarity between the cluster *cid* and all other active clusters and returns the id of *cid*'s nearest neighbor and the similarity between the two clusters. The similarities are computed using the centroids of the clusters.

MergeCentroids creates a new centroid from the centroids of clusters *cid1* and *cid2*. If the weight of a term in centroid i is wt_i, and the number of documents in cluster i is $size_i$, then the weight of the term in the new centroid is

$$\frac{(wt_1 \times size_1) + (wt_2 \times size_2)}{size_1 + size_2}$$

and the size of the new cluster is $size_1 + size_2$. The new centroid contains all the terms in either of the old centroids. If some term occurs in only one of the original centroids, the weight of that term in the other centroid is zero.

```
/* Initialize */
    MaxSim ← 0;
    for (i ← 1 to CollectionSize) {
        create singleton cluster i for doc i;   info[i].centroid ← document i;
        ComputeSim(i, nn, sim);
        info[i].nn ← nn;   info[i].sim ← sim;   info[i].size ← 1;
        if (sim > MaxSim) {
            id1 ←i;   id2 ← nn;   MaxSim ← sim;
        }
    }
    NumActive ← CollectionSize;
    for (i ←1 to NumActive) {
        active[i] ← i;
    }

/* Merge clusters until only one cluster remains or remaining sims are 0 */
    while (MaxSim > 0.0 and NumActive > 1) {
        smaller ← MIN(id1,id2);   larger ← MAX(id1,id2);
        info[smaller].centroid ← MergeCentroids(smaller,larger);
        info[smaller].size ← info[smaller].size + info[larger].size;
        a ← index of larger in active;
        active[a] ← active[NumActive];   NumActive ← NumActive – 1;
        MergeClusters(smaller,larger,MaxSim);
        MaxSim ← 0;
        for (each cluster, a, in active) {
            if (info[a].nn = larger or info[a].nn = smaller) {
                FindMaxSim(a,nn,sim);
                info[a].nn ← nn;   info[a].sim ← sim;
            }
            if (info[a].sim > MaxSim) {
                id1 ← a;   id2 ← info[a].nn;   MaxSim ← info[a].sim;
            }
        }
    }
```

Figure A.3: Constructing the group average link hierarchy

A.1.3. The Complete Link Implementation

The complete link hierarchy construction algorithm has two parts. The first part, shown in Figure A.4, is the procedure that computes and sorts the similarities between documents. The second part, shown in Figure A.5, is the procedure that keeps track of the number of similarities seen between clusters and merges two clusters when all the similarities between documents in those clusters have been seen.

The similarity between documents is computed using the procedure ComputeSims(did). This procedure calculates the similarity between *did* and all other documents in the collection, placing the similarity with document j in the jth element of the global array *sims*. The cosine similarity is computed to three decimal places, implying that there are only 1000 distinct, non-zero similarity values. Therefore, the similarities are sorted using a bucket sort, where each of the 1000 buckets may contain all the pairs of documents that have that similarity.

In the current implementation, the buckets are an array of pointers, indexed by similarity value, where each pointer points to the beginning of a list of fixed length blocks of document pairs. In addition to containing pairs of documents, the blocks contain the similarity value of the documents within the block and a pointer to another block containing document pairs that have the same similarity (if any). That is, the definition of a single block is

{real BlockSim; DOCPAIR pairs[MAX-NUM-PAIRS]; BLOCKPTR next}

and the set of blocks is an array: BLOCK blocks[MAX-NUM-BLOCKS]. The two parameters MAX-NUM-PAIRS and MAX-NUM-BLOCKS control how

```
finished ← FALSE;   MaxSim ← 1.001;
while (not finsihed ) {
    finished ← TRUE;   MinSim ← 0.001;
    for (i ← 1 to CollectionSize) {
        ComputeSims(i);
        for (j ← i + 1 to CollectionSize) {
            if (sims[j] ≧ MinSim and Sims[j] < MaxSim) {
                FoundBlock ← TRUE;
                if (bucket[sims[j]] has no block or all blocks in bucket are full) {
                /* get (another) block for this bucket */
                    MinBlock ← id of block with minimum BlockSim;
                    if (blocks[MinBlock].BlockSim ≠ 0) {
                        finished ← FALSE;
                        if (blocks[MinBlock].BlockSim ≧ sims[j]) {
                            MinSim ← sims[j] + .001;   FoundBlock ← FALSE;
                        }
                        else {
                            MinSim ← blocks[MinBlock].BlockSim + .001;
                            blocks[MinBlock].BlockSim ←sims[j];
                            add block MinBlock to bucket[sims[j]];
                        }
                    }
                }
                if (FoundBlock) {
                    add <i,j> to current block of bucket[sims[j]];
                }
            }
        }
    }
    for (i ← MaxSim – .001 to MinSim by – .001) {
        print contents of bucket[i];
    }
    MaxSim ← MinSim;
}
```

Figure A.4: Bucket sort of similarities for complete link clustering

```
while (EndOfFile ≠ read(x,y,sim)) {
    xid ← map[x];   yid ← map[y];
    if (xid = UNDEF) {              /* item x not seen before */
        NewId ← create singleton cluster for x;   map[x] ← NewId;
        cluster[NewId].list ← NIL;   cluster[NewId].size ← 1;
    }
    if (yid = UNDEF) {              /* item y not seen before */
        NewId ← create singleton cluster for y;   map[y] ← NewId;
        cluster[NewId].list ← NIL;   cluster[NewId].size ← 1;
    }

    if (yid ∉ cluster[xid].list) {
        add yid to cluster[xid].list with count 1;
        add xid to cluster[yid].list with count 1;
    }
    else {
        increment count of yid in xid's list by 1;
        increment count of xid in yid's list by 1;
    }

    /*  merge clusters if this is the last similarity between them  */
    if (cluster[xid].size × cluster[yid].size = count of yid in cluster[xid].list) {
        NewId ← MergeClusters(xid,yid,sim);
        cluster[NewId].size ← cluster[xid].size + cluster[yid].size;
        for (all i ∈ cluster NewId) {
            map[i] ← NewId;
        }
        cluster[NewId].list ← merge(cluster[xid].list, cluster[yid].list);
    }
}
```

Figure A.5: Constructing the complete link hierarchy much memory space is required by the program. Larger values enable the sorting to be performed in fewer iterations. The values in the current implementation are MAX-NUM-PAIRS = 4000 and MAX-NUM-BLOCKS = 500.

The procedure that counts the number of similarities seen between clusters uses lists of [cluster id, similarity count] pairs. The input to the algorithm is the sorted list of [did, did, similarity] triples produced by the previous algorithm. The array *map*, indexed by document id, contains the current active cluster id for each document. When the final similarity is seen between two clusters, procedure MergeClusters(cid1,cid2,sim) is called to merge the two clusters in the hierarchy, and merge(list1,list2) is called to combine the lists of the two clusters into a single list. Merge creates a new list such that any cluster id appearing in exactly one of *list1* or *list2* has the same *count* field in the new list, and the *count* field in the new list of any cluster id in both *list1* and *list2* is the sum of the original *counts*. The list associated with each cluster in *list1* or *list2* is also updated by replacing the old cluster id(s) by the id of the newly merged cluster, and incrementing the *count* field, if necessary.

A.2. Cluster Searches

In this section the four cluster searching algorithms are described. In order to make the algorithms more concrete, the hierarchy is assumed to be implemented as described in section 4.1.2. That is, the hierarchy is represented as a set of nodes where each node contains a node id, a similarity level, a pointer to the vector associated with the node, the number of children the node has, the node id of its first child, and the id of its next sibling. The output of each of the searching algorithms is a sorted array of document ids and similarities called *results*. *NumWanted*, the number of documents to be retrieved, is a parameter of each search. ComputeSim(query,vector) computes the similarity between *query* and *vector*.

A.2.1. Bottom-up Searches

Figures A.6 and A.7 give the algorithms for the bottom-up searches. The search in Figure A.6 retrieves individual documents from within clusters, and the search in Figure A.7 retrieves entire clusters. Both searches begin by performing an inverted index search of the low-level centroids, the result of which is returned in array *top10*. The clusters returned by the inverted index search are then traversed. The search that retrieves individual documents ranks all the documents encountered in the traversal and returns the top *NumWanted* documents. The search that retrieves entire clusters retrieves all documents encountered in the traversal until sufficient documents are retrieved.

A.2.2. Top-down Searches

The top-down searches, given in Figures A.8 and A.9, use a heap to control the search. The search that retrieves individual documents from with in clusters, Figure A.8, pushes and pops triples of the form [node id, sim, type] from the heap. *Type* indicates whether the node represents a cluster or a document. *Sim* is the similarity between the vector associated with the node and the query. The search that retrieves entire clusters, Figure A.9, adds pairs of the form [node id, sim] to the heap. Each search pops items from the heap until sufficient documents are retrieved or the heap is empty.

```
perform an inverted index search of the low-level centroids, returning the
    ids of the ten most similar clusters and the similarities in array top10;
for (i ← 1 to 10) {
    if (top10[i].sim = 0) {
        break;
    }
    if (node top10[i].id has not been visited) {
        GatherChildren(top10[i].id);
    }
}
sort(results);
return(first NumWanted documents);


/* Compute similarity between query and all documents that are
 * descendants of root adding documents to results
 */
GatherChildren(root)
    mark root visited;
    if (node root has no children) {
        sim ← ComputeSim(query, doc vector associated with node root)
        if (sim > 0.0) {
            add node root document's id and sim to results;
        }
    }
    else {                                  /* root represents a cluster */
        for (each child of root) {
            if (child is not visited and (NumWanted + 5) ≧ sizeof(child)) {
                GatherChildren(child);
            }
        }
    }
```

Figure A.6: The bottom-up search that retrieves individual documents

```
perform an inverted index search of the low-level centroids, returning the
    ids of the ten most similar clusters and the similarities in array top10;
MaxSize ← NumWanted + 5;
NumRetrieved ← 0;  NumCentroids ← 0;  i ← 0;
while (i < 10 and top10[i].sim > 0.0 and NumRetrieved < NumWanted) {
    if (node top10[i].id is not visited) {
        GatherDocuments(top10[i].id);
        NumCentroids ← NumCentroids + 1;
    }
    i ← i + 1;
}
sort(results);
return(results);

/* Put documents that are descendants of root into results */
GatherDocuments(root)
    mark root visited;
    for (each child of root) {
        if (child is not visited) {
            if (node child has no children) {  /* found a document */
                NumRetrieved ← NumRetrieved + 1;
                sim ← 1 − (NumCentroids/100);
                add node child document's id and sim to results;
                if (NumRetrieved > MaxSize) {
                    break;
                }
            }
            else {                              /* child represents a subcluster */
                if (MaxSize ≧ NumRetrieved + sizeof(child)) {
                    GatherDocuments(child);
                }
            }
        }
    }
```

Figure A.7: The bottom-up search that retrieves entire clusters

```
PushHeap(root,1,'cluster'); NumRetrieved ← 0;
while (heap is not empty and NumRetrieved < NumWanted) {
    id, sim, type ← PopHeap();
    if (type = 'document') {
        add id and sim to results;
        NumRetrieved ← NumRetrieved + 1;
    }
    else {
        for (each child of node id) {
            sim ← ComputeSim(query,child);
            if (sim ≠ 0.0) {
                PushHeap(child,sim,child-type);
            }
        }
    }
}
sort(results);
return(results);
```

Figure A.8: The top-down search that retrieves individual documents

A.3. The P-norm Inverted Index Search

The last algorithm to be presented is the p-norm inverted index search. Like the other searches, *NumWanted* is a parameter of this search, and the output is the sorted array *results*. The procedure ComputeSim differs slightly from the ComputeSim procedure used with the cluster searches in that this version is passed a list of the weights of the query terms in the document instead of the entire document vector. The algorithm is given in Figure A.10.

```
MaxSize ← NumWanted + 5;   NumRetrieved ← 0;   NumClusters ← 0;
PushHeap(root,1);
while (heap is not empty and NumRetrieved < NumWanted) {
    id, sim ← PopHeap();
    /* cluster is small enough – retrieve it */
    if (MaxSize ≧ sizeof(id) + NumRetrieved ) {
        add all docs in cluster id to results with sim 1 – (NumClusters/100);
        NumRetrieved ← NumRetrieved + sizeof(id);
        NumClusters ← NumClusters + 1;
    }
    /* cluster is too big – add children that are clusters to heap */
    else {
        for (each child of node id) {
            if (node child has no children) {
                sim ← ComputeSim(query,child);
                PushHeap(child,sim);
            }
        }
    }
}
sort(results);
return(results);
```

Figure A.9: The top-down search that retrieves entire clusters

```
bring query terms' inverted lists into core;
ListPointer[i] ← 1;
NextDid ←MaxInt;
for (i ← 1 to number of query terms) {
    if (list[i,ListPointer[i]].id < NextDid) {
        NextDid ← list[i,ListPointer[i]].id;
    }
}

while (some list has a term not yet considered) {
    CurrentDid ← NextDid;   NextDid ← MaxInt;
    for (i ← 1 to number of query terms) {
        if (list[i,ListPointer[i]].id = CurrentDid) {
            weights[i] ← list[i,ListPointer[i]].weight;
            ListPointer[i] ← ListPointer[i] + 1;
        }
        else {
            weights[i] ← 0.0;
        }
        if (list[i,ListPointer[i]].id < NextDid) {
            NextDid ← list[i,ListPointer[i]].id;
        }
    }
    sim ← ComputeSim(query,weights);
    add CurrentDid and sim to results;
}

sort (results);
return(first NumWanted documents);
```

Figure A.10: The inverted index search

Bibliography

[ACM 85]

ACM SIGIR. *Proceedings of the Eighth Annual International Conference,* 1985. ACM order number 606850.

[Anderberg 73]

Michael R. Anderberg. *Cluster Analysis for Applications.* Academic Press, 1973.

[Barraclough 77]

Elizabeth D. Barraclough. On-line searching in information retrieval. *The Journal of Documentation* XXXIII(3):220-238, 1977.

[Buckley 85]

Chris Buckley. Implementation of the SMART information retrieval system. Technical report 85-686, Department of Computer Science, Cornell University. May 1985.

[Chang, Cirrillo & Razon 71]

Y.K. Chang, C. Cirrillo, and J. Razon. Evaluation of feedback retrieval using modified freezing, residual collection, and test and control groups. In [Salton 71a], pages 355-370.

[Croft 78]

W. Bruce Croft. A file organization for cluster-based retrieval. *ACM SIGIR Forum* XIII(1):65-82, 1978.

[Croft 80]

W. Bruce Croft. A model of cluster searching based on classification. *Information Systems* V(3):189-195, 1980.

[Dattola 71]

R.T. Dattola. Experiments with a fast algorithm for automatic classification. In [Salton 71a], pages 265-297.

[Dillon & Desper 80]

Martin Dillon and James Desper. The use of automatic relevance feedback in Boolean retrieval systems. *The Journal of Documentation* XXXVI(3):197-208, 1980.

[Doyle 66]

L. B. Doyle. Breaking the cost barrier in automatic classification. Report SP-2516 System Development Corporation; 2500 Colorado Avenue, Santa Monica, California. July 1966.

[Fox 83a]

Edward Alan Fox. *Extending the Boolean and Vector Space Models of Information Retrieval with P-norm Queries and Multiple Concept Types.* PhD thesis, Cornell University, 1983.

[Fox 83b]

E[dward] A[lan] Fox. 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.

[Grauer & Messier 71]

Robert T. Grauer and Michel Messier. An evaluation of Rocchio's clustering algorithm. In [Salton 71a], pages 243-264.

[Griffiths, Robinson & Willett 84]

Alan Griffiths, Lesley A. Robinson, and Peter Willett. Hierarchic agglomerative clustering methods for automatic document classification. *The Journal of Documentation* XL(3):175-205, 1984.

[Griffiths, Luckhurst & Willett 85]

Alan Griffiths, H. Claire Luckhurst, and Peter Willett, Department of Information Studies, University of Sheffield. Using inter-document similarity information in document retrieval systems. In preparation, September 1985.

[Griffiths & Willett 84]

A[lan] Griffiths and P[eter] Willett. Evaluation of clustering methods for automatic document classification. Report to the British Library Research and Development Department on Project SI/G/564. October 1984.

[Ide 69]

Eleanor Rose Cook Ide. Relevance feedback in an automatic document retrieval system. Master's thesis, Cornell University, 1969. Also Report ISR-15 to the National Science Foundation.

[Jardine & Sibson 68]

N. Jardine and R. Sibson. The construction of hierarchic and non-hierarchic classifications. *The Computer Journal* XI(2):177-184, 1968.

[Jardine & van Rijsbergen 71]

N. Jardine and C. J. van Rijsbergen. The use of hierarchic clustering in information retrieval. *Information Storage and Retrieval* VII(5):217-240, 1971.

[Katzer, *et al.* 82]

J. Katzer, M. J. McGill, J. A. Tessier, W. Frakes, and P. DasGupta. A study of the overlap among document representatives. *Information Technology: Research and Development* I(4):261-274, 1982.

[Kessler 62]

M. M. Kessler. An experimental study of bibliographic coupling between technical papers. Report R-1, Massachusetts Institute of Technology. June 1962.

[Kwok 84]

K. L. Kwok. A document-document similarity measure based on cited titles and probability theory, and its application to relevance feedback retrieval. In C. J. van Rijsbergen, editor, *Research and Development in Information Retrieval*, pages 221-231. Cambridge University Press, 1984.

[Lance & Williams 67]

G. N. Lance and W. T. Williams. A general theory of classificatory strategies: I. Hierarchical systems. *The Computer Journal* IX(4):373-380, 1967.

[Lesser 66]

V. R. Lesser. A modified two-level search algorithm using request clustering. In Report ISR-11 to the National Science Foundation, Cornell University. June 1966.

[Litofsky 69]

Barry Litofsky. *Utility of Automatic Classification Systems for Information Storage and Retrieval.* PhD thesis, University of Pennsylvania, 1969.

[Murray 72]

Daniel McClure Murray. *Document Retrieval Based on Clustered Files.* PhD thesis, Cornell University, 1972. Also Report ISR-20 to the National Science Foundation and to the Nation Library of Medicine.

[Murtagh 83]

F. Murtagh. A survey of recent advances in hierarchical clustering algorithms. *The Computer Journal* XXVI(4):354-359, 1983.

[Murtagh 84a]

F. Murtagh. Complexities of hierarchic clustering algorithms: State of the art. *Computational Statistics Quarterly* I(2):101-113, 1984.

[Murtagh 84b]

F. Murtagh. Structure of hierarchic clusterings: Implications for information retrieval and for multivariate data analysis. *Information Processing and Management* XX(5/6):611-617, 1984.

[Rieber & Marathe 69]

S. Rieber and V. P. Marathe. The single pass clustering method. In Report ISR-16 to the National Science Foundation, Cornell University. September 1969.

[Rocchio 66]

Joseph John Rocchio, Jr. *Document Retrieval Systems – Optimization and Evaluation*. PhD thesis, Harvard University, 1966. Also Report ISR-10 to the National Science Foundation, Cornell University.

[Salton 71a]

Gerard Salton. *The SMART Retrieval System: Experiments in Automatic Document Processing*. Prentice-Hall, 1971.

[Salton 71b]

G. Salton. Cluster search strategies and the optimization of retrieval effectiveness. In [Salton 71a], pages 223-242.

[Salton 75]

Gerard Salton. *Dynamic Information and Library Processing*. Prentice-Hall, 1975.

[Salton, *et al.* 83]

G. Salton, E. A. Fox, C. Buckley, and E. Voorhees. Boolean query formulation with relevance feedback. Technical report 83-539, Department of Computer Science, Cornell University. January 1983.

[Salton, Buckley & Fox 83]

G. Salton, C. Buckley, and E. A. Fox. Automatic query formulations in information retrieval. *Journal of the American Society for Information Science* XXXIV(4):262-280, 1983.

[Salton, Fox & Voorhees 85]

G. Salton, E. A. Fox, and E. Voorhees. Advanced feedback methods in information retrieval. *Journal of the American Society for Information Science* XXXVI(3):200-210, 1985.

[Salton & Voorhees 85]

Gerard Salton and Ellen Voorhees. Automatic assignment of soft Boolean operators. In [ACM 85], pages 54-69.

[Salton & Wong 78]

G. Salton and A. Wong. Generation and search of clustered files. *ACM Transactions on Database Systems* III(4):321-346, 1978.

[Salton, Wu & Fox 83]

Gerard Salton, Edward A. Fox, and Harry Wu. Extended Boolean information retrieval. *Communications of the ACM* XXVI(11):1022-1036, 1983.

[Sibson 73]

R. Sibson. SLINK: An optimally efficient algorithm for the single link cluster method. *The Computer Journal* XVI(1):30-34, 1973.

[Small 73]

Henry Small. Co-citation in the scientific literature: A new measure of the relationship between two documents. *Journal of the American Society for Information Science* XXIV(4):265-269, 1973.

[Smeaton & van Rijsbergen 81]
A. F. Smeaton and C. J. van Rijsbergen. The nearest neighbor problem in information retrieval: An algorithm using upper bounds. *ACM SIGIR Forum* XVI(1):83-87, 1981.

[Tarjan 83]
Robert Endre Tarjan. *Data Structures and Network Algorithms.* Society of Industrial and Applied Mathematics, 1983.

[van Rijsbergen 74]
C. J. van Rijsbergen. Further experiments with hierarchic clustering in document retrieval. *Information Storage and Retrieval* X(1):1-14, 1974.

[van Rijsbergen & Croft 75]
C. J. van Rijsbergen and W. B. Croft. Document clustering: An evaluation of some experiments with the Cranfield 1400 collection. *Information Processing and Management* XI(5-7):171-182, 1975.

[van Rijsbergen & Sparck Jones 73]
C. J. van Rijsbergen and K. Sparck Jones. A test for the separation of relevant and non-relevant documents in experimental retrieval collections. *The Journal of Documentation* XXIX(3):251-257, 1973.

[Voorhees 85]
Ellen M. Voorhees. The cluster hypothesis revisited. In [ACM 85], pages 188-196.

[Willett 84]

Peter Willett. A note on the use of nearest neighbors for implementing single linkage document classifications. *Journal of the American Society for Information Science* XXXV(3):149-152, 1984.

[Williamson 74]

Robert Edward Williamson. *Real-time Document Retrieval.* PhD thesis, Cornell University, 1974.

[Worona 71]

S. Worona. Query clustering in a large document space. In [Salton 71a], pages 298-310.

[Wu 81]

Harry Chih Chien Wu. *On Query Formulation in Information Retrieval.* PhD thesis, Cornell University, 1981.

[Yu 73]

Clement Tak Yu. *Theory of Indexing and Classification.* PhD thesis, Cornell University, 1973.