Implementing Agglomerative Hierarchic Clustering Algorithms for Use in Document Retrieval*

Ellen M. Voorhees

TR 86-765
July 1986

Department of Computer Science
Cornell University
Ithaca, NY 14853

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

Implementing Agglomerative Hierarchic Clustering Algorithms for Use in Document Retrieval*

Ellen M. Voorhees

Cornell University

Abstract: Searching hierarchically clustered document collections can be effective [6], but creating the cluster hierarchies is expensive since there are both many documents and many terms. However, the information in the document-term matrix is sparse: documents are usually indexed by relatively few terms. This paper describes the implementations of three agglomerative hierarchic clustering algorithms that exploit this sparsity so that collections much larger than the algorithms' worst case running times would suggest can be clustered. The implementations described in the paper have been used to cluster a collection of 12,000 documents.

For a retrieval system to be useful, the component that selects the documents that are to be returned to the user must be both effective and efficient. Jardine and van Rijsbergen suggested the use of agglomerative hierarchic clustering as a means of increasing the effectiveness and efficiency of searches as compared to a sequential scan of the collection [1]. Subsequent research has confirmed that searches of some agglomerative hierarchies can be more effective than a total ranking of the document collection by decreasing similarity to the query [2,3,4,5,6].

Unfortunately, creating the cluster hierarchies is expensive. Algorithms that consider the similarity between all pairs of items have at least an $O(N^2)$ time dependency, where N is the number of items to be clustered, and many of the clustering methods have an $O(N^2)$ space dependency as well. Algorithms which avoid looking at all inter-item similarities by finding nearest neighbors in other ways assume the items are described by a very small number of attributes (see [7] for a discussion of these methods). In information retrieval, both the number of items to be clustered (documents) and the total number of attributes describing the items (terms) are typically over 10,000.

Although the total number of documents and terms is large, the information is sparse: any given document is described by relatively few terms and has a non-zero similarity with relatively few other documents. Thus some of the algorithms whose worst case running times are intimidating may be feasible in practice. For example, Croft created a single link hierarchy for a collection of 11,613 documents using an inverted index of the collection to avoid computing zero similarities [8]. Willett and El-Hamdouchi created Ward's cluster hierarchies by adapting a nearest neighbor algorithm [9].

This paper describes the implementation of single link, complete link and group average link clustering algorithms that have been used to cluster collections of up to 12,000 documents [6]. The single link program has a running time of $O(N^2)$ and a space requirement of $O(N)$. (Sibson gives

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

another single link clustering algorithm requiring $O(N^2)$ time and $O(N)$ space [10].) The average link algorithm computes the group average link hierarchy for a restricted set of similarity measures: the mean similarity between documents in two sets S_1 and S_2 must equal the similarity between the mean document of S_1 and the mean document of S_2. Using this restriction, the algorithm constructs the hierarchy in $O(N^2)$ time and $O(N)$ space – no general algorithm for creating the group average or complete link hierarchies that uses less than $O(N^2)$ space while using only $O(N^2)$ time is known [11]. The complete link algorithm takes $O(N^3)$ time and $O(N^2)$ space, but requires much less than N^2 space in practice due to the relatively few non-zero similarities.

1. Definition of the Clustering Methods

Before describing the implementation of the various algorithms, the different clustering methods should be defined. Single link, complete link, and group average link clustering are members of a class of clustering methods known as agglomerative hierarchic clustering. All of the members of this class are defined similarly:

1. Each item to be clustered is a singleton cluster.
2. While there is more than one cluster, the clusters with the maximum similarity are merged (ties are broken arbitrarily) and the similarity between the newly merged cluster and the remaining clusters is recomputed.

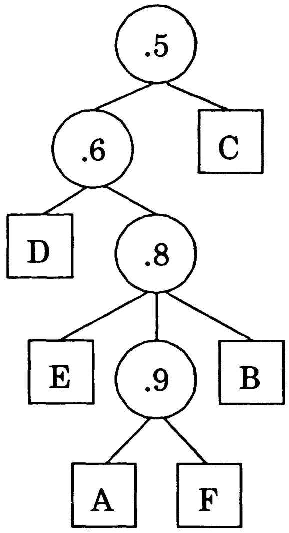
The resulting clustered collection consists of a hierarchy of items in which small clusters of very strongly related items are nested within larger clusters of less strongly related items. (Equivalent definitions of the clustering methods for distance measures are obtained by substituting "distance" for "similarity" and interchanging "minimum" and "maximum".)

The difference among the methods within the class of agglomerative hierarchic clustering is the way in which 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.

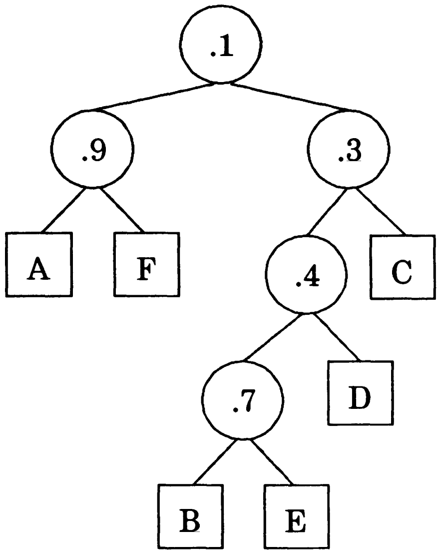
The hierarchies defined by each of these methods for a sample set of data are given in Figure 1. In the Figure the hierarchies are represented by trees in which the leaves represent documents and the interior nodes represent non-singleton clusters. The similarity level at which a cluster forms is recorded in the node that represents the cluster. All the documents that are descendants of an interior node belong to the cluster represented by that node.

	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

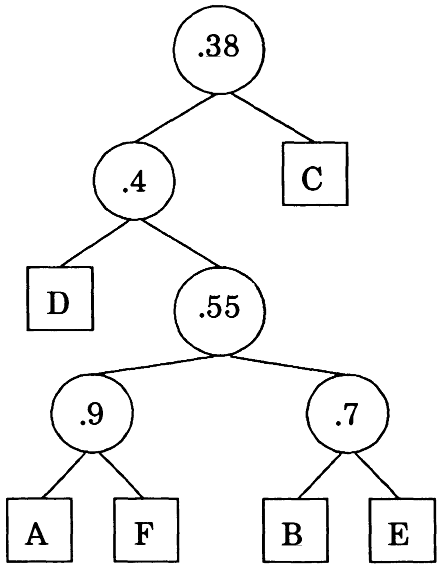
a) the similarity matrix



b) the single link hierarchy



c) a complete link hierarchy



d) a group average link hierarchy

Figure 1: Cluster hierarchies for a sample set of data

Equivalent alternative definitions of the single link and complete link methods can be found in the literature [12]. 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

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
NIL	null pointer

Figure 2: Notation used in algorithms

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. Implementation of Clustering Methods

The implementation of the three clustering methods is given in this section. In addition to giving a high-level description of the algorithms in English, an example construction and a description using an Algol-like pseudo-language are given. Figure 2 contains the notation used in the psuedo-code description. No particular representation of the cluster hierarchy is assumed. It is assumed that procedures for inserting a document into the hierarchy at a given similarity level and for merging two clusters at a given similarity level while retaining the structure of the clusters being merged are available.

2.1. 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 [13]. 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 1. If the algorithm begins by placing document A in the hierarchy, the initial state of the algorithm is as depicted in Figure 3a. The maximum similarity between a document in the hierarchy and a document not in the hierarchy is between documents A and F, so document F is added to the hierarchy and the array of similarities is updated (Figure 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 3c). In each subsequent iteration, the document with the maximum similarity to a document in the hierarchy is added to the hierarchy (Figure 3d and Figure 3e). The algorithm terminates when the last document is added to the hierarchy (Figure 3f).

The code for the single link method is in Figure 4. 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 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*.

2.2 The Group Average Link Implementation

It was mentioned in the introduction that no general algorithm is known for computing the group average link and the complete link hierarchies in $O(N^2)$ time while using less than $O(N^2)$ space. The difference in the time and space complexity of these methods on the one hand and the single link method on the other is the amount of information that needs to be stored to compute the similarities between non-singleton clusters. For single link clustering, merging two clusters can never decrease the similarity between clusters, and thus only the cluster with maximum similarity to a given cluster needs to be stored for the given 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 re-computed when needed (which makes the running time greater than $O(N^2)$), or they need to be stored (which takes $O(N^2)$ space).

The group average link hierarchy can be constructed in $O(N^2)$ time and $O(N)$ space when the similarity between documents is equivalent to the inner product of two vectors using appropriately weighted vectors. 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

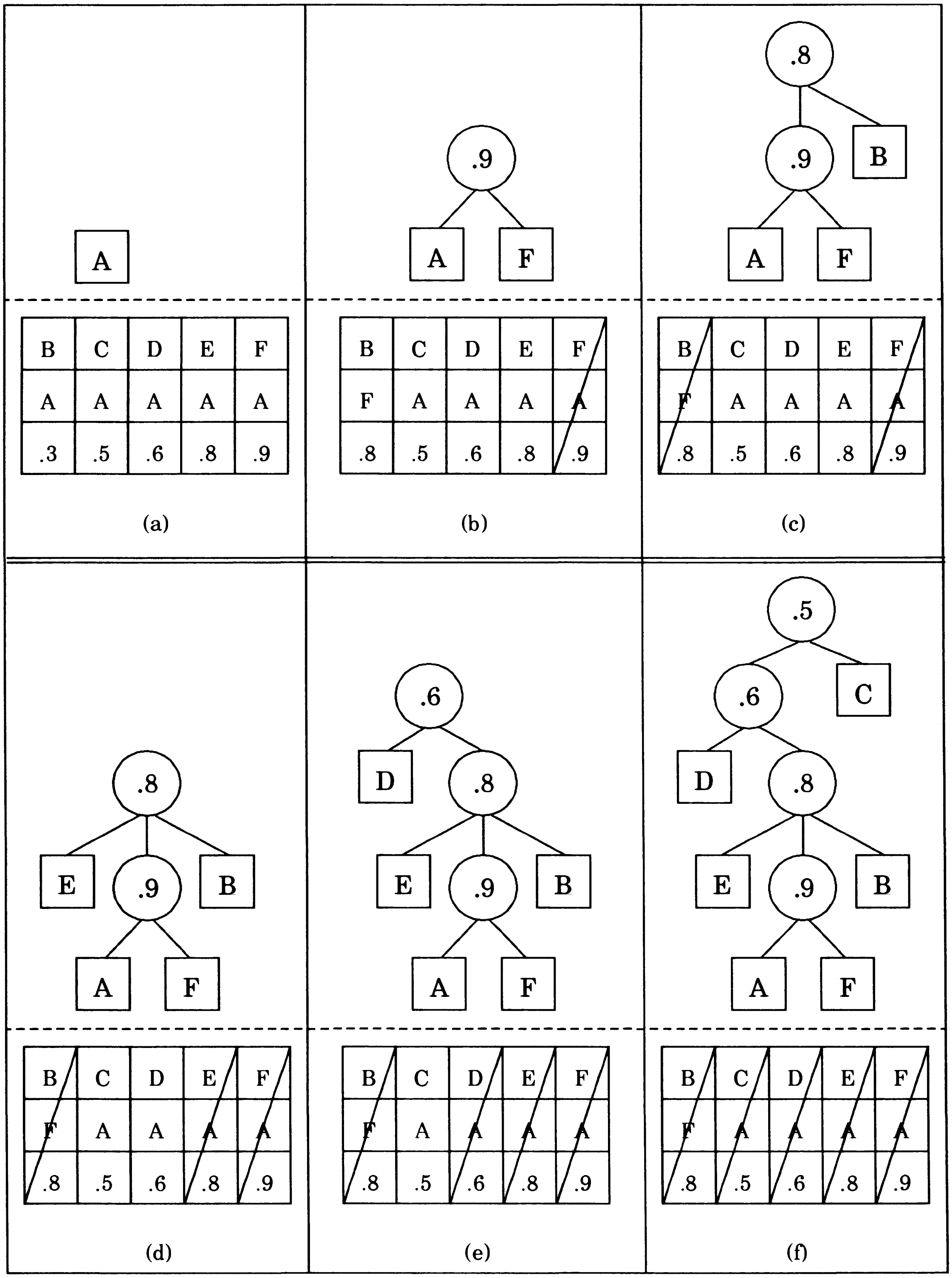


Figure 3: Example of constructing the single link hierarchy

```
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 4: Constructing the single link hierarchy

centroids of the cluster can be used to compute the similarities between the clusters while requiring only O(N) space. The current group average link implementation uses centroids as cluster representatives in an adaptation of Anderberg's stored data approach algorithm [12, p. 146].

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 5 contains an example of constructing a group average link hierarchy using the algorithm described above and the collection of Figure 1. 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 5a; 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 5b. 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 in the example, although the new most similar cluster and corresponding similarity are shown.

After the second iteration (Figure 5c), 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 5f).

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.

The group average link procedure is given in Figure 6. 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).

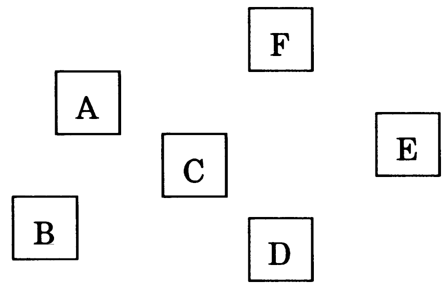
ComputeSim calculates the similarity between document *did* and the other documents in the collection. It returns the id of the document with maximum similarity with *did* in *nn* (nearest neighbor) and the similarity between the two documents in *sim*. The current implementation of this procedure uses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

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

CLUSTER	1	2	3	4	5	6
MOST SIM	6	6	5	1	1	1
SIMILARITY	.9	.8	.5	.6	.8	.9
MEMBERS	A	B	C	D	E	F

merge clusters 1 and 6

(a)

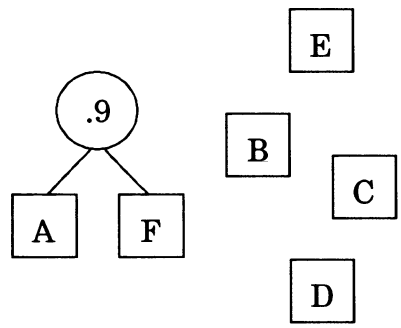


ACTIVE: 1, 2, 3, 4, 5

CLUSTER	1	2	3	4	5
MOST SIM	2	5	5	2	2
SIMILARITY	.55	.7	.5	.5	.7
MEMBERS	A,F	B	C	D	E

merge clusters 2 and 5

(b)



ACTIVE: 1, 2, 3, 4

CLUSTER	1	2	3	4
MOST SIM	2	1	2	2
SIMILARITY	.55	.55	.45	.45
MEMBERS	A,F	B,E	C	D

merge clusters 1 and 2

(c)

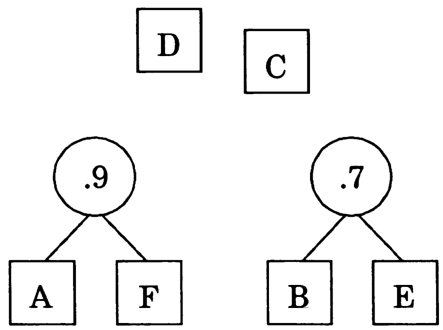


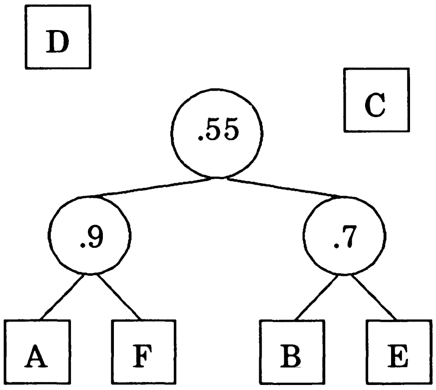
Figure 5: Example of constructing a group average link hierachy

ACTIVE: 1, 3, 4

CLUSTER	1	3	4
MOST SIM	4	1	1
SIMILARITY	.4	.4	.4
MEMBERS	A,B,E,F	C	D

merge clusters 1 and 4

(d)

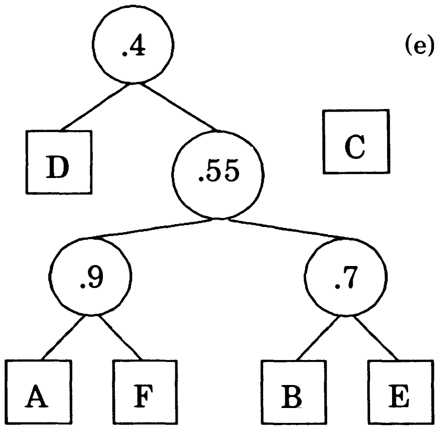


ACTIVE: 1, 3

CLUSTER	1	3
MOST SIM	3	1
SIMILARITY	.38	.38
SIZE	A,B,D,E,F	C

merge clusters 1 and 3

(e)



ACTIVE: 1

(f)

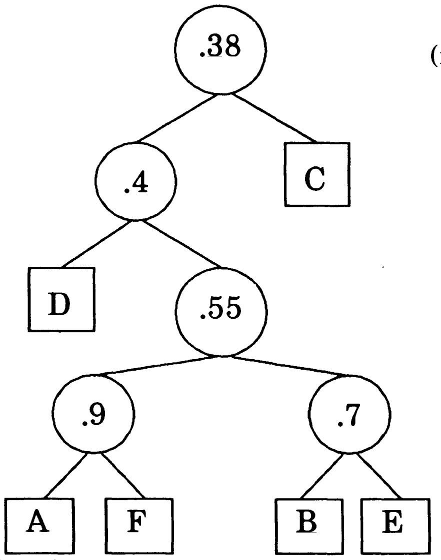


Figure 5 (continued)

```
/* 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 6: Constructing the group average link hierarchy

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 size of the new cluster is $size_1 + size_2$ and the weight of the term in the new centroid is

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

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.

2.3. 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 1 is shown in Figure 7. In the example, three is used for the value of k in phase one of the algorithm.

Figure 7a 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

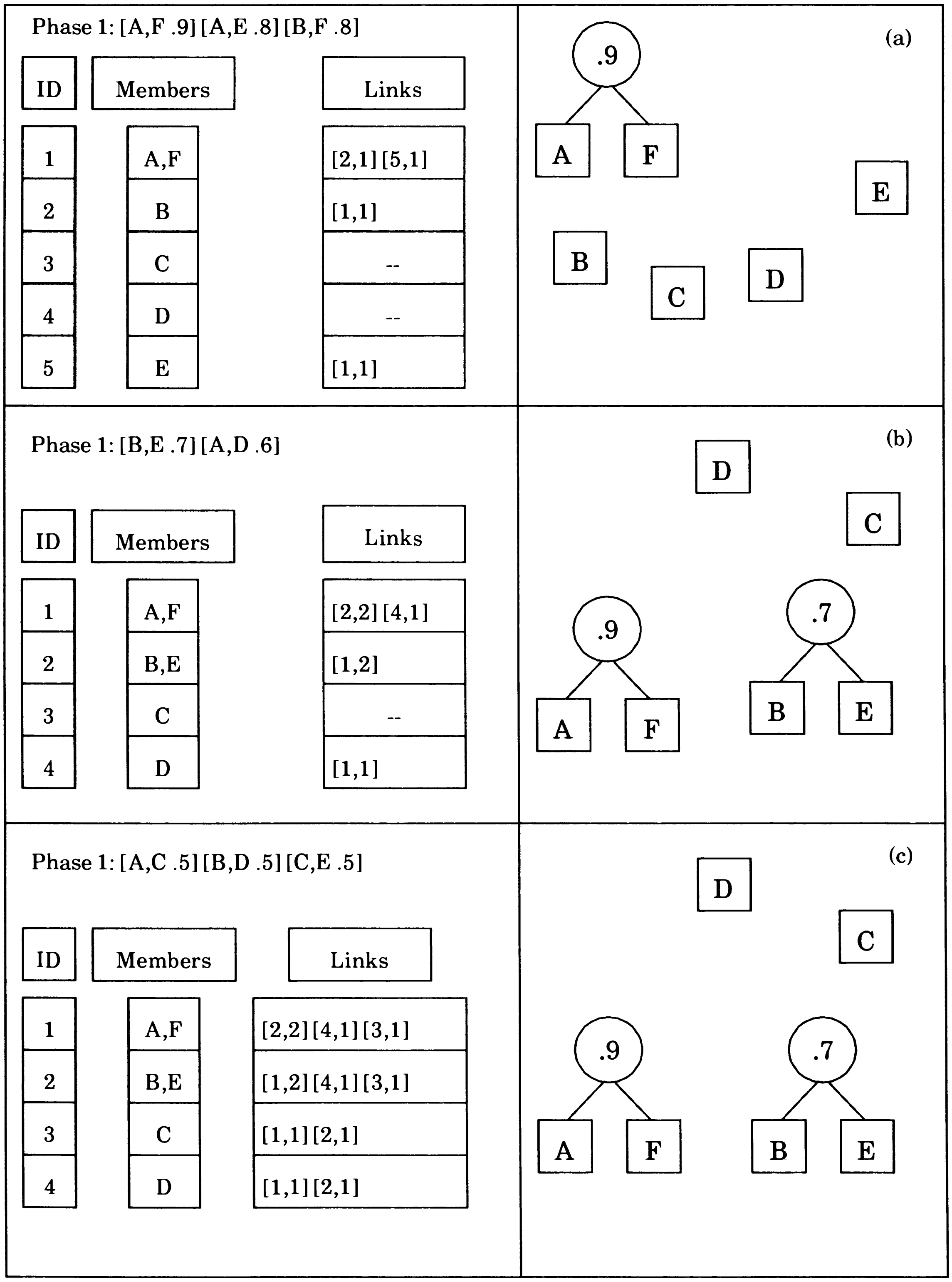
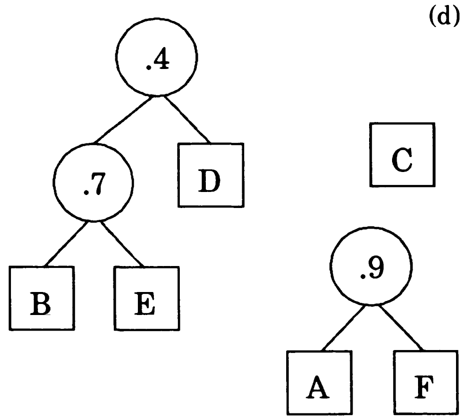


Figure 7: Example of constructing a complete link hierachy

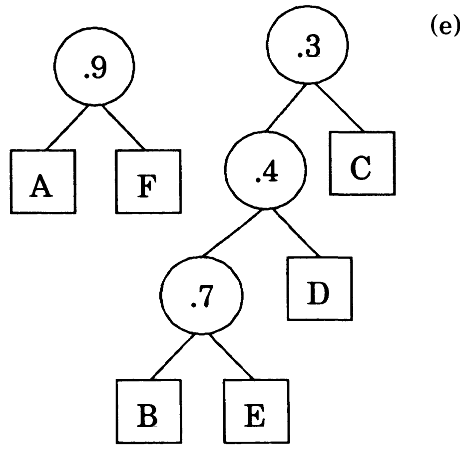
Phase 1: [D,E .4] [B,C .4]

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



Phase 1: [D,F .1]

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

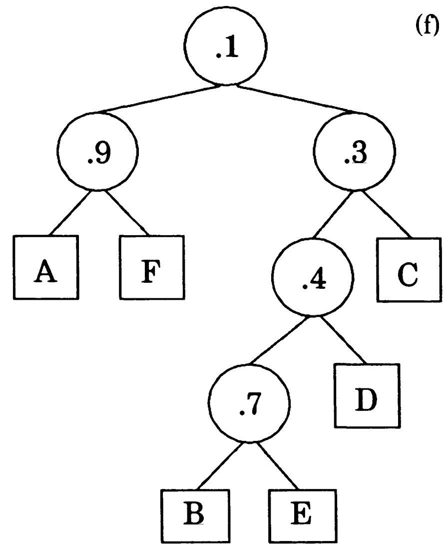


Figure 7 (continued)

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 7b, 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 7c, do not cause any mergers. The first similarity of the iteration depicted in Figure 7d, 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 7e, and the final two clusters merge in the last iteration. The algorithm terminates after the last similarity ([D,F .1]) has been processed.

The two parts of the complete link clustering algorithm are shown in Figures 8 and 9. The first part, shown in Figure 8, is the procedure that computes and sorts the similarities between documents. The second part, shown in Figure 9, 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 similarity is computed to three decimal places, and since the similarity lies in the range $0 \leq sim \leq 1$, this implies 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, each bucket is an element of an array of pointers, indexed by similarity value, such that 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 much memory space is required by the program.

```
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 8: Bucket sort of similarities for complete link clustering

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

```
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 9: Constructing the complete link hierarchy

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.

3. Experience

The implementations described above were used to cluster collections containing up to 12,684 documents for a study of the usefulness of agglomerative hierarchic clustering in information retrieval [6]. The programs were specifically designed to work in conjunction with the SMART information retrieval system [14]. They were written in C and run under UNIX™ on a VAX™ 780 with a floating point accelerator and six megabytes of memory.

The clustering programs needed to be changed as the larger collections were clustered so the clustering time would remain feasible (the implementations given above are the final versions). Unfortunately, this means no accurate performance statistics are available. Several modifications were caused by the necessity of keeping disk I/O to an absolute minimum. For example, in the current implementation, each program has a subprogram that computes the similarity between documents. In the original versions, the similarities were input to the programs, which increased the programs' modularity and portability, but caused the performance to suffer intolerably.

The amount of physical memory available is of critical importance for the same reason: there must be sufficient memory to contain the program's working set so it can avoid substantial paging. The six megabytes was ample memory for the group average link and single link programs on the 12,684 document collection, but was insufficient during the early stages of the complete link program.

Using the implementations described in this paper, group average link and complete link hierarchies were constructed for document collections larger than those that have been clustered with these methods before now. These programs can be used on collections of this size because they specifically exploit the fact that a given document has a non-zero similarity with relatively few other documents. Thus fewer than the theoretically necessary $N(N-1)/2$ inter-document similarities need to be computed, and, in the complete link program, fewer than $O(N^2)$ links between clusters need to be stored.

Although the collections are large when compared to the usual size of sets of objects to be clustered using the complete link or group average link methods, they are still fairly small when compared to the size of document collections used in operational retrieval systems. In order for agglomerative hierarchic methods to be useful in larger scale retrieval systems, methods that inexpensively approximate the hierarchies in such a way as the hierarchies can still be searched effectively are needed. 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 [15]. A similar approximation of the group average link and complete link hierarchies may also be effective.

References

[1] Jardine, N.; van Rijsbergen, C. J. "The Use of Hierarchic Clustering in Information Retrieval." *Information Storage and Retrieval* 7(5): 217-240; 1971.

[2] van Rijsbergen, C. J. "Further Experiments with Hierarchic Clustering in Document Retrieval." *Information Storage and Retrieval* 10(1): 1-14; 1974.

[3] van Rijsbergen, C. J.; Croft,W. B. "Document Clustering: An Evaluation of Some Experiments with the Cranfield 1400 Collection." *Information Processing and Management* 11(5-7): 171-182; 1975.

[4] Croft, W. B. "A File Organization for Cluster-Based Retrieval." *ACM SIGIR Forum* 13(1): 65-82; 1978.

[5] Croft, W. B. "A Model of Cluster Searching Based on Classification." *Information Systems* 5(3): 189-195; 1980.

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

[7] Murtagh, F. "A Survey of Recent Advances in Hierarchical Clustering Algorithms. *The Computer Journal* 26(4): 354-359; 1983.

[8] Croft, W. B. "Clustering Large Files of Documents Using the Single-Link Method." *Journal of the American Society for Information Science* 28(6): 341-344; 1977.

[9] El-Hamdouchi, A.; Willett, P. "Hierarchic Document Classification Using Ward's Clustering Method." In preparation. January 1986.

[10] Sibson, R. "SLINK: An Optimally Efficient Algorithm for the Single Link Cluster Method." *The Computer Journal*. 16(1): 30-34; 1973.

[11] Murtagh, F. "Complexities of Hierarchic Clustering Algorithms: State of the Art." *Computational Statistics Quarterly*. 1(2): 101-113; 1984.

[12] Anderberg, M. R. *Cluster Analysis for Applications*. New York: Academic Press; 1973.

[13] Tarjan, R. E. *Data Structures and Network Algorithms*. Society of Industrial and Applied Mathematics; 1983.

[14] Salton, G.; McGill, M. J. *Introduction to Modern Information Retrieval*,Chapter 4. New York: McGraw-Hill; 1983.

[15] Willett, P. "A Note on the Use of Nearest Neighbors for Implementing Single Linkage Document Classifications." *Journal of the American Society for Information Science*. 35(3): 149-152; 1984.