GENERATION AND SEARCH OF CLUSTERED FILES

G. Salton, D. Bergmark, and A. Wong

TR 77-299

Department of Computer Science
Cornell University
Ithaca, N.Y. 14853

Generation and Search of Clustered Files

G. Salton, D. Bergmark, and A. Wong *

Abstract

A classified, or clustered file is one where related, or similar records are grouped into classes, or clusters of items in such a way that all items within a cluster are jointly retrievable. Clustered files exhibit substantial advantages in many retrieval environments over the more conventional inverted list or multilist technologies.

An inexpensive file clustering method applicable to large files is given together with appropriate file search methods. An abstract model is used to predict the retrieval effectiveness of various search methods in a clustered file environment, and experimental evidence is introduced to confirm the usefulness of the model. As an example, a collection of research papers in computer science is clustered automatically, and the resulting research clusters are compared with existing, manually constructed taxonomies for the computer field.

1. Conventional Retrieval Environments

In the computer science literature the information retrieval field is separated into two unequal portions: by far the largest part is devoted to single-key searches, that is, to retrieval activities where only a single key is used at any one time for search purposes; a small

---

*Department of Computer Science, Cornell University, Ithaca, New York, 14853.

This study was supported in part by the National Science Foundation under grant GJ 43505.

part of the literature — probably less than ten percent of the output — concerns itself with the so-called secondary key, or multi-key retrieval methods. The reasons for the emphasis on single-key searches are two-fold. First, certain well-known processes studied by computer scientists make it necessary to perform searches based on single search arguments — this is notably the case for the symbol table searches used in compiling, and for a variety of dictionary look-up processes. Second, single-key searches are now well understood; a great many sophisticated data structures and search methodologies have been devised, including refined hashing methods and tree search processes, leading to optimal single-key searches under specified conditions. [1]

By comparison, retrieval problems based on multiple keys have been neglected, in part, one may suspect, because the mathematical constructs that are often used for single keys do not extend easily to the more general situation: effective key-address (hash) transformations are difficult to construct when a large number of keys must be simultaneously manipulated, and the well-known search tree construction methods produce large, unwieldy constructs when a given tree node represents a multiplicity of keys. [2,3]

Unfortunately, it is often impossible in practice to formulate one's search requirements in terms of single search arguments, or even as sequences of single arguments. In the business world, for example, one may wish to identify all employees exhibiting certain job classifications, who also fall within certain age brackets, and possess certain specified skills. Alternatively, one may wish to extract from a library file all items falling into a number of specified subject areas.

In such circumstances, one is forced to revert to secondary, multi-key retrieval methods and to the few well-known file organization and search methods which apply. When the file size is small, or when rapid responses are not essential, a sequential search may be made through an unorganized file by successively comparing the given search arguments with the identifiers attached to the various stored records. Thus the first record is compared with the query, then the second record, and so on, until finally the last record is treated. On average, one-half of the stored records must be examined to identify any particular wanted record.

When the response time or file size requirements make it impossible to use sequential searches, an auxiliary index, or directory is normally used which may identify for each key all the record numbers or record addresses in the main data file containing the given key; alternatively, the directory may identify only the first record containing each key, and this record in turn may lead to subsequent records for the same key. In the former case, one speaks of a so-called inverted file organization; in the latter case, the organization is known as a multi-list file.

In an inverted file system, a search is conducted in several steps [4,5]:

a) For each term included in the search request the corresponding list of record numbers or record addresses is obtained from the directory.

b) These record lists are combined, or merged to produce all record numbers or addresses that satisfy the query specification.

c) The actual data records corresponding to the previously determined record numbers are retrieved from the main file.

In a multi-list file, only the first record pertaining to a given key is identified in the directory, and the records exhibiting a common search key are chained together in the main file, that is, connected by a chain of pointers. Retrieval then takes place by identifying in the directory the shortest record list (pointer chain) for any of the keys included in the query statement. The records in that chain are then individually examined before retrieval to determine whether the complete search requirements are actually satisfied in each case. An idealized diagram of the inverted and multi-list file search strategies is contained in Fig. 1.

An inverted file system leads to rapid retrieval because the list processing operations involving the directory (steps (a) and (b) above) are performed in fast storage, whereas retrieval from the main data file is restricted to those records only that actually correspond to the full query statement. Retrieval from a multi-list file may also be reasonably rapid provided that the number of records corresponding to the various query keys is small, that is, provided the length of the record chains in the main file is limited. It can be shown that for short record lists the multi-list system is generally more efficient than the inverted file process; the reverse obtains when the number of records corresponding to a given search key is large. [6]

Unfortunately, both the multi-list and the inverted file systems exhibit two substantial disadvantages:

a) Any search key actually used in practice must be provided for in the directory in advance; for practical purposes this implies that the search vocabulary (index terms, key words, etc.) used to formulate the search requests is closed and prespecified; furthermore each search request must be formulated completely and precisely using the preestablished system vocabulary.

b) Because the retrieval strategy is based on an exact match between search keys included in the query statements and identifiers attached to the stored records, retrieval is an "all or nothing" proposition, in that all items that completely match the search specification are retrieved whereas all others are rejected; furthermore, the user has little control over the size of the output set, and no obvious method suggests itself for ranking the output in some presumed order of potential usefulness.

In actual fact, the conditions under which retrieval activities are ideally carried out are quite different: much of the time, the user does not know how to formulate a precise, unambiguous query, but can instead suggest tentative, incomplete statements to be refined during the course of the search action; in the same way, the stored records might not be completely specifiable because the actual content of an item may be in doubt, or because ambiguous record attributes may in fact be appropriate in some cases. This suggests that the use of complete, unambiguous record and query specifications based on static, preconstructed vocabularies may be unnatural.

What is wanted instead is a system where approximate searches can be conducted between partially formulated queries and possibly incompletely specified information items leading to the identification of a few potentially relevant items. When such an item proves useful upon further examination it should be possible to "zero-in" on the corresponding

portion of the stored record collection so as to retrieve additional items similar to those previously identified as potentially interesting. Such a retrieval environment appears to lead directly to the use of clustered files.

2. Clustered File Organizations

A clustered file organization is one where similar, or related records are grouped into classes, or clusters of items in such a way that all items within a common cluster are jointly accessible without excessive delay. The place of a given record within the stored file depends on the similarity of that record with other records in the file. Normally this similarity between file items is ascertained by performing a global comparison of the record identifiers attached to the corresponding items.

Consider as an example a record identified by t attributes (or keys, identifiers, index terms, etc.). Such a record may be represented by a t-dimensional vector $D_i = (d_{i1}, d_{i2}, \ldots, d_{it})$ where d_{ij} represents the weight, or importance of the jth identifier attached to record D_i. If each stored record is a vector in t-space, the complete file becomes a t-dimentional vector space. The similarity between pairs of items may then be computed as a vector function between the corresponding attribute or term vectors. Typical similarity functions might be the inner product g of the term vectors, or the cosine h of the angle formed by the vectors in t-space. Specifically,

$$g(D_i,D_j) = \sum_{k=1}^{t} d_{ik}\, d_{jk}$$

and

$$h(D_i,D_j) = \frac{g(D_i,D_j)}{\sqrt{g(D_i,D_i)\cdot g(D_j,D_j)}} .$$

Based on the similarity concept between records it is now easy to define a clustered file as one which groups into common classes, or clusters, those items whose pairwise similarity is sufficiently large. A typical clustered file organization is shown in Fig. 2, where each x represents a record, and the circular configurations are the clusters. The distance between two x's in the two-dimensional representation of Fig. 2 is assumed inversely related to the similarity between the corresponding records, or rather between the term vectors representing the records. Thus, when the clusters are far removed from each other, the records will exhibit substantial differences. For overlapping clusters, on the other hand, the corresponding records may present considerable similarity.

In the diagram of Fig. 2 each record cluster is identified by a dummy central item known as the centroid. Typically the centroid of a cluster is another t-dimensional vector, computed for example as the vector sum of all records included in the cluster. In a clustered file organization, a search is carried out by first comparing a query formulation with the cluster centroids. This may then be followed by a comparison between the query and those individual records whose corresponding query-centroid similarity was found to be sufficiently large in the earlier comparison. It is clear that the store of cluster

centroids carries out in a clustered file environment the same function as the keyword directory for an inverted or multi-list file, in that the centroids are used to provide access to some of the stored records. [7,8]

The foregoing introduction indicates that clustered file searches can be rapidly conducted because large portions of the file are immediately rejected, the search being concentrated in areas where substantial similarities are detectable between queries and cluster centroids. Furthermore, records similar to a given sample record, or to a given query, are easily identified because a complete cluster of items is normally stored in adjacent storage locations (for example, on the same track of a disc file), and hence becomes available simultaneously. Since the retrieval process depends on a global match between the complete query formulation and the content identifiers attached to the records, the operations will carry through also for partially, or incompletely specified items.

The major problem relating to clustered files arises not in connection with the search operation but rather with the file generation process. This problem is examined in more detail in the next few paragraphs.

3. Cluster File Generation and Search

The vast majority of the automatic classification methods actually used in practice are based on the availability of a complete similarity matrix specifying the similarity between all pairs of records. In such a case, s_{ij}, the matrix element at the intersection of row i and column j of the matrix represents the similarity coefficient between records D_i and D_j. By suitable transformations of the rows and columns of the

similarity matrix it may be possible to collect many of the large similarity coefficients (matrix elements) in certain areas of the similanity matrix. This in turn leads to the identification of clusters of related items exhibiting large pairwise similarities. [8,9,10]

Unfortunately, the well-known standard classification methods which follow this scheme all exhibit a complexity of at least order n^2, since for n records n(n-1)/2 similarity coefficients must be computed for the generation of the similarity matrix alone. Such a clustering process then becomes too expensive for practical implementation with large files. In practice, it is necessary to resort to the esthetically less satisfying single-pass clustering methods in which each item is processed only once, a pairwise similarity matrix is not required, and the cost of generation is of order n log n for n items. [7,8]

Typically the construction process is bottom-up. The first item is initially identified with cluster one. The next item is compared with cluster one and merged with it if found to be sufficiently similar. If the new item is not similar to any already existing cluster, a new cluster is generated. Whenever a new item is entered into a cluster, the corresponding cluster centroid must be redefined by incorporating terms from the new term vector into the original cluster centroid.

In principle, such a process should serve to assign each item to at least one cluster, and the classification should be complete after one pass through the file. In practice, the resulting classes may not be usable for search purposes without additional refinements. Several problems arise:

a) the number of clusters produced by the one-pass system may become excessively large, implying that a query submitted to the system may have to be matched against a very large number of centroids before access to the individual records is actually obtained;

b) the size of certain clusters may become too large, particularly if a great many records in a collection cover a fairly homogeneous subject area;

c) alternatively, the cluster size may be very small, and could indeed be limited to a single record in cases where so-called "loose" records exist that do not match any other records in the collection;

d) the overlap among clusters, that is, the number of items jointly contained in more than one cluster may be too large or too small.

To respond to such eventualities, controls must be introduced to regulate cluster size, cluster overlap, and number of clusters generated. Furthermore, special provisions must be made for the loose items that do not properly relate to any of the existing clusters. [11,12] Concerning first the loose items, a really efficient solution appears to be lacking. As an ad-hoc measure, such items could remain unclustered, particularly if relatively few such items exist. Alternatively, loose items might be merged with the closest existing clusters, that is, with clusters with which the item correlates most highly; or finally loose items might be collected into special clusters for which the normal affinity requirements between cluster elements are relaxed.

Reasonable solutions do exist to control size and number of clusters. Thus when a cluster becomes too large during cluster generation, that is when it contains too many individual records, its centroid may be split, and two new clusters may emerge from a single original one. Such a splitting operation can be initiated automatically whenever the cluster

sizes exceed a preestablished threshold. Unfortunately, as more centroids are created, the search operation becomes less and less efficient, since many query-centroid comparisons must then be made before any of the records can actually be retrieved. This suggests that a hierarchy of clusters be created: first the individual records are grouped into clusters; the cluster centroids defining these low-level clusters may themselves be grouped into superclusters defined by supercentroids. If too many superclusters should exist, the corresponding supercentroids may be collected into larger hyperclusters, identified by hypercentroids, and so on. The centroid splitting operation must then operate equally at each level of the cluster hierarchy: whenever any grouping (cluster, supercluster, hypercluster, etc.) contains too many substructures (records, centroids, supercentroids, etc.), a split may be made to create two smaller groups in place of a large one.

A sample cluster splitting operation is shown in Fig. 3 where the assumption is made that no grouping may contain more than four subelements. The initial state consists of four clusters, each containing between two and four records. The cluster centroids are labelled A, B, C, and D, respectively. In Fig. 3(a), a dummy supercentroid, labelled S, identifies a single supercluster containing the four original centroids. If a new record is added to cluster A, an illegal situation arises since the cluster size is assumed limited to four elements. The A centroid may then be split thereby creating two new centroids A' and A'' as shown in Fig. 3(c). At this point the supercluster S is no longer viable since it now contains five subelements. This is remedied by splitting S into S' and S'' and creating a dummy hypercluster labelled H in Fig. 3(d).

It will be recognized that this cluster maintenance process is in fact identical to the bucket splitting process used when too many keys congregate in a single bucket of a B-tree. [12,13,14] The B-tree operations for reductions in file size also carry through in that it may then become necessary to delete one or more clusters by merging two centroids into one.

A typical hierarchical cluster structure obtained from a single-pass cluster generation process is shown in Fig. 4(a). The structure consists of 33 records grouped into 11 clusters. The 11 cluster centroids are themselves arranged in four superclusters, and these in turn appear in two large hyperclusters. The search tree corresponding to Fig. 4(a) is shown at the bottom of Fig. 4. An incoming user query, or a new record to be added to the file is first compared with the two hypercentroids (labelled 1 and 2 in Fig. 4(b)). Depending on the magnitude of the similarity coefficient between the input item and the hypercentroids, a comparison next occurs with supercentroids 3 and 4, or with supercentroids 5 and 6. The supercentroids in turn lead to the third level centroids and eventually to the individual records themselves.

A simplified flowchart of the single-pass cluster generation and search process is shown in Fig. 5. Generation and search differ in substance only for the lowest level centroids: during cluster generation a new record must be added on the lowest level of the cluster tree and the cluster splitting routine may need to be invoked; for searching, on the other hand, the low-level clusters simply lead to lower level individual records.

When cluster splitting is in order, the program of Fig. 5 simply inserts the corresponding cluster identifier onto a list of items to be split. The splitting routine itself is represented in detail in the chart of Fig. 6. The programs of Figs. 5 and 6 refer to the "nodes" of a cluster tree, to the "father" of a node on the next higher level, and the "sons" of a node on the next lower level. These terms are standard designations in normal tree processing algorithms.

Clearly the search strategy may be adapted to individual user requirements by suitably adjusting the thresholds of the various similarity coefficients as one proceeds through the cluster structure. Narrow, depth-first searches may thus be conducted by using at each level only the highest matching item to reach a lower level. Narrow searches may be expected to produce high retrieval precision and low recall.* Alternatively, broader searches may be performed by developing (that is, by comparing with the incoming query) several highly correlating elements at each level of the tree. In that case the recall may be higher but the precision may suffer. It is not hard to show that for the single-pass process previously outlined, the number of vector comparisons (between queries, incoming new records, centroids, etc.) needed for cluster searching, cluster generation, and cluster tree updating is of order n log n for a file of n records. [12]

---

*Precision is defined as the proportion of retrieved items actually found relevant, whereas recall is the proportion of relevant materials actually retrieved. When the recall is high most of the items relevant to a given user query will have been retrieved successfully. High precision, on the other hand implies that much of the extraneous material will have been rejected. It may happen on occasion that everything of interest is retrieved while all extraneous items are rejected. In that case both the recall and the precision measures will attain maximum values of 1.

Since cluster processing is effectively implemented as a tree tracing process, these complexity bounds are not likely to be lowered any further.

4. A Cluster Search Model

The cluster search strategy of Fig. 5 provides an overall framework for traversing the cluster hierarchy leading to an eventual comparison between the incoming query and some of the records on the lowest level of the cluster structure. Unfortunately the search strategy itself provides no clue about the individual parameter values to be used in a given search. In particular, it is not clear how many clusters should be examined at each level of the cluster hierarchy in order to obtain a specified number of desired records, nor is any information provided about the threshold values to be used for the similarity computations between individual centroid or record vectors.

These questions may be studied by examining a simple probabilistic model of cluster searching which is capable of predicting under well-defined conditions the expected number of desired records contained in each cluster of a clustered file. Given such a model, it is possible to fix in advance the various parameter values which must be used to ensure the retrieval of a given number i of desired records. [15]

Consider, in particular, a query Q of length ℓ (that is, containing ℓ attributes), and a cluster containing m records. Both the query and the record attributes are assumed to be binary, that is, q_{ij} (or d_{ij}) is equal to 1 whenever the jth attribute is assigned to query Q_i (or to record D_i), and is otherwise equal to 0. Furthermore, let the similarity

between queries and stored records be measured as the inner product between the corresponding attribute vectors, that is, $g(Q_i,D_j) = \sum_{k=1}^{\ell} q_{ik}\, d_{jk}$. Since the vectors are assumed to be binary, this function simply measures the number of common attributes between the two vectors.

Two additional assumptions are made for present purposes: first, the query attributes are assumed to be independently assigned to the records within a given cluster; and second, no overlap is assumed to exist between clusters, that is, a given record is assigned to at most one low-level cluster. The first assumption appears reasonable because the records within a common cluster are necessarily related in subject matter. The second assumption is in fact unnecessary; however the model becomes too complicated for present purposes when cluster overlap is included. [15]

Let y_j represent the number of records in a cluster of m records containing query attribute j. Then y_j/m is the probability that a random record in the cluster contains the jth query attribute. Since the query attributes occur independently in the records, the probability that a random record contains exactly the k query attributes $j_1,j_2,\ldots,j_k$, but not the ℓ-k attributes $j_{k+1},\ldots,j_\ell$ becomes

$$\left(\prod_{p=1}^{k} y_{j_p}/m\right)\left(\prod_{p=k+1}^{\ell} [1-y_{j_p}/m]\right).$$

Since there are $\binom{\ell}{k}$ ways of choosing k attributes out of ℓ, the probability that a random record in cluster r contains <u>any</u> k properties in common with a query of length ℓ is

$$P_r(k) = \sum_{(k)} \left(\prod_{p=1}^{k} y_{j_p}/m \right) \left(\prod_{p=k+1}^{\ell} [1-y_{j_p}/m] \right), \qquad (1)$$

and the expected number of records out of m records in cluster r having exactly k properties in common with query Q is

$$C_r(k) = m \sum_{(k)} \left(\prod_{p=1}^{k} y_{j_p}/m \right) \left(\prod_{p=k+1}^{\ell} [1-y_{j_p}/m] \right).$$

Finally, the expected number of records in cluster r having at least k properties in common with query Q is

$$E_r(k) = \sum_{p=k}^{\ell} C_r(p). \qquad (2)$$

By computing the E value for the various file clusters and assuming that a record containing a sufficient number of matching query attributes is in fact relevant to that query, it becomes possible to devise a reasonable cluster search strategy. Let i be the total number of records to be retrieved in a given search, and let $\Delta \geq 0$ be a constant such any cluster containing Δ or fewer expected number of desired records will not be included in the search effort (because the expected search pay-off would be too small in such a case). An appropriate search strategy is then as follows:

1) Retrieve records from clusters for which the expected number of desired records is greater than Δ for each cluster, that is

$$\sum_{p=k}^{\ell} C_r(p) > \Delta \quad (r=1,\ldots,n).$$

2. Since the aggregate number of records to be retrieved must be greater than i for properly chosen k, the added condition must be

$$\sum_{r} \sum_{p=k}^{t} C_r(p) \geq i.$$

When overlap exists among the clusters, it becomes necessary to compute the expected number of records having at least k properties in common with the query that are situated in the intersection between two or more adjacent clusters. The overlapping items must then be subtracted to reach an accurate E value. When the overlap is small, the calculated E value of equation (2) may however be expected to hold even in the more complicated situation. [15]

The foregoing search algorithm will prove useful when the calculated expected number of useful records is close to the actual number of records in the cluster identified as relevant by the user submitting the original search request. This correspondence may depend on how closely the assumptions of the model are actually satisfied in practice. In particular, when the query terms are not independent of each other but are instead semantically related, the calculated E value may substantially differ from the real one. This suggests that the model may be more appropriate for short queries with few attributes than for longer queries where the independence assumptions are more questionable.

To confirm the reasonableness of the model, E values were calculated for a number of actual clusters and search requests using a collection of documents (records) in aerodynamics together with queries submitted by a user population of researchers in the aeronautics field. In each case, a cluster-query pair was chosen and the actual occurrence

probabilities y_{j_p}/m were used for each of ℓ query terms to compute an E value as a function of k, the number of matching query-document attributes.

Typical output results (E values) are shown in Table 1 for 3 different clusters and two different queries, including a long query of 12 terms and a short one of 5 terms. The Table also shows the actual number of relevant and nonrelevant documents included in the respective clusters and exhibiting the appropriate number of query term matches (the appropriate k values). Considering part (a) of Table 1, it is seen that cluster 38 contains 7 documents in all, of which 6 are identified as relevant by the user submitting query 1. The nonrelevant item has 2 attributes in common with the query. Of the six relevant ones, two exhibit two query terms, two more include three query terms, one has four terms in common with the query, and the last one has five common query terms. It may be seen that the computed E values are very close to the actual values in that case. When all items are retrieved with at least one matching query term, the E value predicts 7 desired documents and 6 actually exist; for at least two matching terms the E value goes down to 6.9 while the actual number of useful records remains at 6. Finally, when items with at least 3 matching terms are retrieved, E becomes 4.9 and the actual number of retrieved useful items is 4.

An examination of the remaining output of Table 1 indicates that when k is small, that is when a great many items are retrieved, the predicted value is not very close to the actual one. The reason is that

many stored records may be expected to exhibit one or two term matches in common with the query even though the records may not be relevant to the corresponding query. However, as the number of term matches needed for retrieval increases, the estimated probability value E becomes very close to the actual one. For k=3 or larger, the calculated values shown in Table 1 appear to be very good indeed. One concludes that the model is adequate for practical utilization.

5. Cluster Search Strategies

The probabilistic model examined in the previous section may be tested by using sample document collections with actual user queries in a clustered retrieval environment. A clustered collection of 424 documents in aerodynamics is therefore used experimentally with 24 user queries; the recall-precision output is averaged in each case over all 24 queries. [16]

The following principal cluster search strategies suggest themselves:

a) A standard similarity computation between the query and the various cluster centroids can be used to identify the n clusters with the highest query-centroid values; the items in these clusters can then be compared with the query and recall-precision values can be obtained after all documents in the n best clusters are processed. For present purposes, the cosine measure h is used to compare queries with cluster centroids and documents. This standard cluster search process may be expected to produce a high-level of performance when the user requirements are fairly homogeneous, because variations in the search strategy are not required in that case from one query to another.

b) Instead of using a standard query-centroid similarity computation, it is possible to perform the probability calculations of equation (2) to obtain for each cluster and for fixed values of k the expected number of records in the cluster having at least k matching attributes in common with the query. All clusters with E values higher than a given threshold can then be compared with the query before recall and precision values are computed. Obviously when k is small, large values of E are obtained producing high recall output but low precision because many nonrelevant items may be expected to be retrieved together with many relevant ones. As k grows, few clusters will exhibit large E values and the search pattern may favor high precision and lower recall values.

The $E_r(k)$ measure depends on cluster size. An alternative probability measure independent of the number of records per cluster is the probability that a random record in cluster r contains at least k properties in common with the query. From equation (1) this P-value is defined as $\sum_{p=k}^{t} P_r(p)$.

c) The process based on the probability function may be rendered more flexible by using a varying value of k for the individual queries. Thus for each query, the value of k is incremented until equation (1) shows that a random record in the third highest ranked cluster has a probability smaller than 1 of exhibiting exactly k matching terms in common with the query. Contrariwise, the value of k is decreased by 1 if the probability value of a random record in the highest ranked cluster is smaller than 0.3. In effect, the varying k method chooses a particular k value for each query in such a way that the number of clusters to be examined is restricted to only those which exhibit a high relevance probability of containing useful material. A high k value is used for queries producing large E values for many clusters, whereas low values of k are produced when the E values are generally low.

d) An alternative approach to the use of varying values of k consists in looking at the variance of the distribution obtained by plotting the P-values $\sum_{p=k}^{t} P_r(p)$ for the various document clusters of the collection and for specific values of k. In each case, the clusters may be arranged in decreasing order of their P values for a given value of k and a given query, and these P values may be plotted along the ordinates of a two-dimensional plot against the cluster numbers shown along the abscissa. A monotonically decreasing curve results which becomes more peaked as the value of k increases. The variance method consists in using for each query a normalized distribution of the P-values (for which the area under the P curve is the same in each case) and in picking that value of k for which the variance of the normalized distribution is smaller than a given threshold. An idealized picture of the variance changes with increasing k threshold is shown in Fig. 7.

The effect of the variance process is the same as that of the varying k in that a large k is obtained for queries producing large P values for many clusters; smaller k values are used for queries for which this is not the case.

Two kinds of evaluation output may be exhibited. The first shows variations in recall and precision output as more and more document clusters are compared with the user queries; the other compares the retrieval effectiveness for a fixed number of clusters examined using the various search strategies described earlier. The output shown in Table 2 exhibits recall-precision values as a function of the number of expanded document clusters for a collection of 424 documents in aerodynamics, averaged over 24 user queries. Four search methods are used including the standard cosine similarity comparison, the probability measure for

two fixed values of k, and the varying k method.

It is clear from the output of Table 2 that as expected the recall will improve as more and more clusters are examined, and the precision suffers accordingly. For the cosine function the identical search strategy can always be applied to all 24 queries. This is not the case, however, for the probability computations: for a fixed value of k and increasing numbers of clusters examined, the required probability values are obtained for fewer and fewer queries; the problem worsens as k increases. Thus for k=4, there are only 20 queries out of 24 for which it is possible to find a single cluster with nonzero probability of having a random record exhibit four matching query terms. When 18 or 20 clusters must be found with the required nonzero probabilities, only 9 queries out of 24 can be used for the recall-precision computations. The right-hand column of Table 2(d) shows that many more queries can be used with the varying k method than for fixed k.

Table 2 demonstrates that the standard cosine measure produces high recall values when many clusters are examined; when very few clusters are looked at, the one or two most important clusters are easier to find using the probability calculations. The varying k method represents a good compromise between cosine and fixed k, because the recall-precision values approach the fixed k output when few clusters are used and the cosine when many clusters are expanded. The best retrieval precision is clearly obtained for large values of the threshold k. In that case, only the most relevant clusters are actually matched with the query; unfortunately not many queries exist for which such high standards of relevance actually exist.

To obtain an accurate picture of retrieval system performence, it is appropriate to perform a detailed comparison of the search methods for some fixed average number of expanded clusters. Table 3 shows the number of expanded clusters averaged over 24 queries for various search strategies involving fixed k, varying k and variance methods. The figures of Table 3 apply to a situation where the maximum number of clusters expanded for a given query is either 7 or 20, and two different thresholds are used in the E values to decide upon the total number of clusters to be looked at in each case. In the first case, labelled 7-1 and 20-1 in Table 3, no further clusters are expanded after finding one for which the expected number of relevant records (that is, the expected number of records with the requisite number of matching attributes between query and documents) is less than 1. In the two other cases, labelled 7-3 and 20-3 respectively, no further clusters are expanded upon finding a cluster for which the expected number of relevant records is less than 3.

In Table 4, average precision values are shown for fixed values of the recall. A number of search strategies are compared, the output being grouped so that the average number of expanded clusters is the same within each group. Thus in Table 4(a) approximately 3 clusters are expanded on average for the 24 queries. The average number of clusters expanded increases to 5, 7, and 14 in Tables 4(b), 4(c), and 4(d) respectively. The actual search strategy used is listed under each column of precision values: the maximum number of expanded clusters i, and the stopping criterion for further consideration of additional clusters j is designated as i-j, as previously explained. The exact number of clusters actually used on average for the 24 queries is shown

in parentheses under the respective column. Within each subsection of Table 4, the best precision values are identified by a vertical bar.

Once again higher performance values are obtained as more clusters are expanded. Obviously the better performance is achieved at the cost of more search effort. For a given common level of search effort, the standard cosine process is again best at the high recall end of the spectrum. The probability computations are most useful when high retrieval precision is wanted; of the various search strategies which use the probability measure, the varying k or variance procedures are preferred over the fixed k methods. The fixed k process should be used with large k values only when outstanding performance for a few queries is preferred over a reasonable average performance in all queries.

6. Document Clusters in Computer Science

The cluster file concept is best understood by considering a cluster structure involving actual records. In principle, it is possible to produce clusters for ordinary business records, such as records of employees in a personnel file, or customer accounts in a bank. In such business environments, it may be possible to implement both a flexible cluster search system based on a clustered file system, as well as the usual exact match capability currently used in mechanized data base retrieval environments. The latter could be obtained, for example, by adding to the normal clustered data file an inverted directory constructed from the cluster centroid terms. [7,8]

For tutorial purposes, it is however convenient to use as a clustering example a file in which the individual records represent the documents of a particular collection of books or articles. In

that case, the individual document clusters are similar in concept to the subject classes with which one deals routinely in a normal library environment. Under normal circumstances, the subject headings or other content identifiers attached to the individual documents might constitute the principal clustering criterion used to produce the document clusters; that is, items exhibiting similar subject descriptions would be grouped into clusters.

In the present study, the content identifiers or index terms used to control the clustering process are replaced by references and citations. A reference is a bibliographic item appearing in the bibliography of a document (A refers to B); a citation, on the other hand, is a reference made by an outside item to a given document (B cites A). The clustering criterion to be used operates in such a way that the similarity coefficient between two items will depend on the number of references shared by their bibliographies, and on the number of common citations from the outside.

Consider, as an example, a given document $D = (c_1, c_2, \ldots, c_n, r_1, r_2, \ldots, r_m)$ where c_i represents the ith citation, and r_j the jth reference. When two items share many citations, the c-terms in the respective document vectors exhibit many common elements; hence the corresponding similarity coefficient between these two items will be large. The same is true for items sharing many common references (many common r-terms).

As an example of the clustering process, a hierarchical multi-level clustered file was generated for a collection of research articles in computer science. The base collection consisted of 334 articles appearing

either in the ACM literature during 1974 (ACM Communications, ACM Journal, and ACM Computing Surveys), or in Computer and Control Abstracts, an abstract journal published by Inspec in England, during 1968 and 1971. Articles published in two different time periods were chosen for the base collection in the hope of obtaining a number of cross-citations from one base article to another. In addition to the base collection of 334 articles all documents appearing in the lists of bibliographic references attached to each of the base documents were included in the collection. These reference lists contained an average of 11 references per paper, or about 3670 articles. After elimination of all duplicates, the completed collection contained 3,520 distinct articles, including the 334 base papers.

The complete document collection may be further broken down into the following subsets:

a) 3,186 distinct articles not included in the base collection that were referred to by one or more of the base articles but did not themselves cite other base articles in the collection;

b) 33 base articles that were referred to by other base articles but did not themselves cite other base articles in the collection;

c) 13 base articles referred to by other base articles that also cited one or more other base article; and

d) 30 base articles that were not themselves referred to by base articles but that cited one or more base article.

A summary of the collection make-up is contained in Fig. 8. If a link is defined as either a reference or a citation from one article to another, the 3,520 distinct collection items used for the experiment generate 3,893 distinct links.

The bottom-up clustering process of Fig. 5 can be used to produce document clusters consisting of groups of related research articles. Because of the method followed in building the experimental collection, two slight modifications were made in the basic procedure. First, a preliminary pass through the collection was used to merge with the respective citing items the document vectors of all items cited by a single document only that did not themselves cite other collection items. In other words, initial clusters were formed which would contain a given citing item plus all its singly cited items that did not themselves cite anything else in the collection. Furthermore, the items identified as loose during the cluster generation process were not forced into other clusters, but were instead kept unclustered. This would make it possible to determine what type of documents in a homogeneous subject area would prove difficult to classify automatically.

A summary of the cluster generation process for the 3520 computer science documents is shown in Fig. 9 and the corresponding cluster statistics are contained in Table 5. Fig. 1 and Table 5 show that the initial pass (pass 1 or P1) combining a given citing item with its singly cited references produces 327 clusters of about 9.5 documents per cluster. 415 of the original 3520 documents remain unclustered after pass 1. Of the 327 pass-1 clusters, 74 were "complete" after pass 1 in the sense that these clusters were divorced from all other elements in the structure (the similarity between any of these 74 clusters and the remaining centroids or documents of the collection was zero). Such complete clusters remain unchanged for the remainder of the clustering process. The 74 completed pass-1 clusters comprising 490 documents are shown separately in Fig. 9.

A standard clustering pass is now used to produce the pass-2 clusters. A total of 61 pass-2 clusters are generated in pass 2 containing an average of 33 documents per cluster. Seven of the 61 pass-2 clusters were complete after pass-2 and cannot be absorbed with other elements of the collection. After pass 2, 121 original documents are still loose, and 166 pass-1 clusters still remain in the system, including of course the 74 completed ones. Fig. 9 shows that 22 pass-3 clusters are eventually generated comprising an average of 95 documents per cluster, and 9 pass-4 clusters containing approximately 275 documents each.

At the conclusion of the clustering run, the entire structure consists of 12 initial documents which could not fit into any of the groupings. In addition, 91 pass-1 clusters still remain, including the 74 complete ones; 8 pass-2 clusters including the 7 complete ones; two pass-3 clusters; and finally 9 large pass-3 clusters. Failure to insist on absorbing the poorly matching constructs (documents or clusters) into higher level structures produces an unbalanced search tree shown in Fig. 10, where some items are reachable more quickly (with a smaller number of comparison operations) than others. Thus the 42 pass-2 clusters on level 3 of the tree are reached by first matching a search request with the 9 centroids for pass-4 clusters, followed by a comparison with 20 pass-3 centroids. On the other hand, the 8 pass-2 clusters which are never absorbed into larger clusters are available without any intermediate comparisons. The tree of Fig. 10 can be balanced by forcing all items into upper level constructs on the next higher level of the search tree, or alternatively by defining dummy centroids to replace the intermediate nodes left empty in the tree of Fig. 10.

7. Automatic Clustering and Computer Science Taxonomies

It is impossible in the present context to provide a detailed analysis of all the individual clusters of research papers generated by the clustering process. It may, however, be worth comparing a few of the automatically generated research clusters with some of the existing computer science taxonomies. This could determine affinities between recognized subtopics of the field and actual research output. The pass-1 clusters consisting of reference lists attached to individual documents are too small for this purpose, and the pass-3 clusters containing on average almost 100 documents are too big. The 61 pass-2 clusters with an average size of 33 documents per cluster may best fit the subtopics in the established taxonomies.

Before proceeding with such a comparison, it should be remembered that only about 2000 of the 3520 original documents are actually included in a pass-2 cluster. Thus not all active research areas are reflected in the P2 subset; however, each of the 61 clusters should be recognizable in the various taxonomies. Three well-known computer science taxonomies may be used for present purposes, including the well-known classification designed for the ACM Computing Reviews (CR) [17], the taxonomy introduced by the NSF sponsored Computer Science and Engineering Research Study (Cosers) [18], and finally the subject classification used to arrange articles in the recently published Encyclopedia of Computer Science [19]. Table 6 shows a list of the major topic classes that define the three classification systems, as well as a subcategory count for each class.

An attempt to apportion the 61 pass-2 research clusters to the subcategories of the various taxonomies raises a number of problems in each case. By far the most troublesome questions arise for the Computing

Reviews taxonomy. Over half of the research clusters fall into a single topic class (Mathematics of Computation), and two other major classes (Functions, and Analog Computers) are vacuous, or nearly so. The Cosers classification represents a great improvement over CR, but there are some highly idiosyncratic features — for example, the important "Applications" area is vacuous for Cosers even though there is significant research coverage in the applications sections of the competing classifications, and the usefulness of the "Symbol Processing" topic class is not obvious. By far the most suitable and the most comprehensive classification is the one proposed in the Encyclopedia.

A detailed list of the problems which arise in fitting each of the 61 research clusters to the hundreds of subtopic classes defined for the three classification systems cannot be given here. A summary must suffice, starting with the CR classification. The following major problems are apparent:

a) the hardware area is split up into two major topic classes for no apparent reason; the "analog computer" class is vacuous of research clusters and its existence appears unnecessary as a separate major topic area;

b) the software area, on the other hand may be too large as currently constituted; the topics related to programming languages and programming practice could be split off from the software management area as is done in the Encyclopedia classification;

c) the applications class is divided by CR into areas of ultimate use, such as natural sciences, engineering, humanities, and so on; a much more felicitous arrangement for computer science purposes would be a subdivision according to the technique being used for solving a problem, or by aspect or property of a problem;

d) the topic class entitled "Functions" appears to be a mixture of application topics such as graphics, and mathematical techniques, such as simulation methods, and operations research;

e) the mathematics class includes both numerical methods and theory of computation, thus accounting for over half of the research clusters under a single heading;

f) each major topic class includes two subclasses entitled "general" and "miscellaneous", respectively; these subclasses understandably do not fit any recognizable research topic;

g) several active topics appear to be lost for practical purposes: for example, classification theory, security-privacy, theorem proving, microprocessing, radix conversion, and others.

The Cosers classification presents fewer problems than CR. Still a few questions of substance must be raised:

a) the applications area includes nine special topics, such as, for example, weather prediction, moon orbit calculations, and computer animation; excluded are many other equally deserving topics, with the result that its matching coefficient with the 61 research clusters is a perfect zero;

b) a special major topic class is assigned to artificial intelligence, but not to any other major area of applications techniques;

c) a curious area entitled "symbol processing" and recently renamed "special topics" falls squarely between applications and theory; included in this class are list processing and algebraic manipulations, but also text processing; the latter topic is shown under "special topics" and a second time in the data management class; the whole special topics class has a flavor similar to that suggested by the "miscellaneous" categories defined for CR;

d) quite a few active topics exist that don't find a ready home in the Cosers taxonomy, including for example, network topology, sorting, information retrieval, social implications, and others.

As indicated earlier, the Encyclopedia classification presents the fewest problems. There seem to be no extraneous major categories, and the problems that do arise may be common to all computer science taxonomies:

a) in the applications area, it is difficult to distinguish between computational techniques common to many applications, and specific methods applied to a given field only;

b) the line between mathematical techniques and theory is hard to draw; for example, in the Encyclopedia, automata theory appears under mathematics but formal languages belongs to theory;

c) data management which is given a separate heading by Cosers appears under computer systems in the Encyclopedia where it is thus bracketted with computer networks and time-sharing;

d) some deserving topics are not explicitly mentioned, such as, for example combinatorial methods which should be included under theory, or possibly under mathematics for computing; and logic which could go under many headings, including also applications where theorem proving now appears.

The foregoing analysis makes it clear that none of the existing taxonomies will provide a perfect fit for actually existing research collections in the field. However the difficulties which arise in fitting

the research clusters to the classification systems are easy to discover, and the basic usefulness of any given manually constructed scheme becomes obvious. Furthermore, the automatic cluster structure can be used to suggest improvements in the manual classifications and to obtain a better understanding of a given field of endeavor.

One may expect that when the construction and use of clustered files becomes more widespread in the future, automatic clustering procedures will suggest themselves not only for file organization, search, and retrieval purposes, but also for the analysis and identification of influential contributors and important topic classes in a given discipline.

References

[1] D.E. Knuth, The Art of Programming, Vol. 3, Sorting and Searching, Addison Wesley Publishing Company, Reading, Mass., 1973.

[2] J.C. Bentley, Multidimensional Binary Search Trees Used for Associative Searching, ACM Communications, Vol. 18, No. 9, September 1975, p. 509-516.

[3] R.A. Finkel and J.L. Bentley, Quad Trees — A Data Structure for Retrieval on Composite Keys, Acta Informatica, Vol. 4, No. 1, 1974, p. 1-9.

[4] I. Flores, Data Structure and Management, Prentice Hall Inc., Englewood Cliffs, N.J., 1970.

[5] D. Lefkovitz, File Structures for On-line Systems, Spartan Books, New York, 1969.

[6] C.S. Yang, Directory Design and Record Allocation for List and Cluster Files, Cornell University Ph.D. Thesis, Ithaca, New York, March 1976.

[7] G. Salton, editor, The SMART System — Experiments in Automatic Document Processing, Prentice Hall Inc., Englewood Cliffs, N.J., 1971.

[8] G. Salton, Dynamic Information and Library Processing, Prentice Hall Inc., Englewood Cliffs, N.J., 1975.

[9] J.A. Hartigan, Clustering Algorithms, John Wiley and Sons, New York, 1975.

[10] R.R. Sokal and P.H.A. Sneath, Principles of Numerical Taxonomy, H.P. Freeman and Sons, San Francisco, 1963.

[11] D.B. Johnson and J.M. Lafuente, A Controlled Single Pass Classification Algorithm with Application to Multi-level Clustering, Scientific Report No. ISR-18, Section 12, Computer Science Dept., Cornell University, Ithaca, N.Y., October 1970.

[12] R.E. Williamson, Real Time Document Retrieval, Cornell University Ph.D. Thesis, Ithaca, New York, 1974.

[13] R. Bayer and E. McCreight, Organization and Maintenance of Large Ordered Indices, Acta Informatica, Vol. 1, No. 3, 1972, p. 173-189.

[14] R. Bayer, Symmetric Binary B-trees: Data Structure and Maintenance Algorithms, Acta Informatica, Vol. 1, No. 4, 1972, p. 290-306.

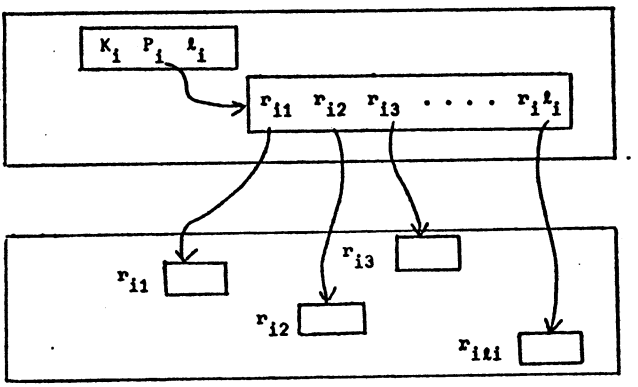
[15] C.T. Yu, W.S. Luk, and M.K. Siu, On the Estimation of the Number of Desired Records With Respect to a Given Query, Technical Report, Computing Science Department, University of Alberta, Edmonton, Alberta, 1976.

[16] G. Salton, C.S. Yang, and C.T. Yu. A Theory of Term Importance in Automatic Text Analysis, Journal of the ASIS, Vol. 26, No. 1, January-February 1975, p. 33-44.

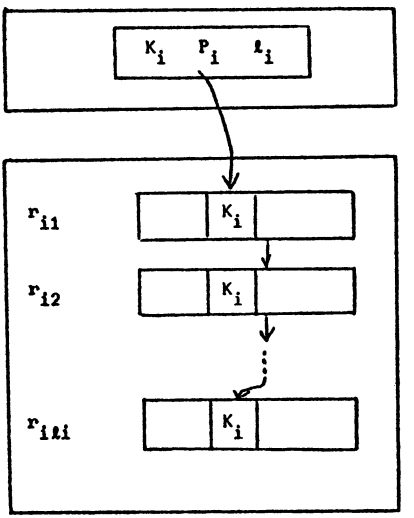
[17] Categories of the Computing Sciences, Classification System for Computing Reviews — A Description, Computing Reviews, Vol. 17, No. 5, May 1976, p. 175-198.

[18] B.W. Arden, The Computer Science and Engineering Research Study (Cosers), ACM Communications, Vol. 19, No. 12, December 1976, p. 670-673.

[19] A. Ralston and C.L. Meek, Encyclopedia of Computer Science, Petrocelli/Charter Publishing Co., New York, 1976.



a) Inverted List Organization

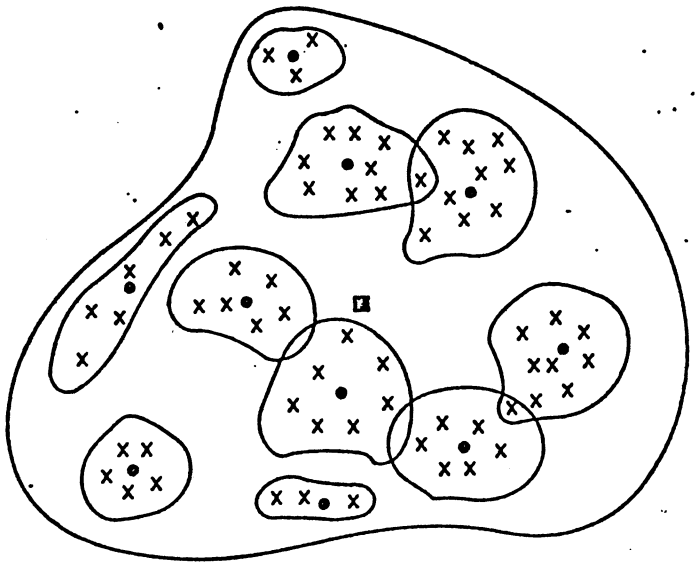


b) Multilist Organization

K_i	key i
i	number of r for key K_i
r_{ij}	address for record unde K_i
P_i	pointer for K_i

Inverted and Multilist File Organizations

Fig. 1



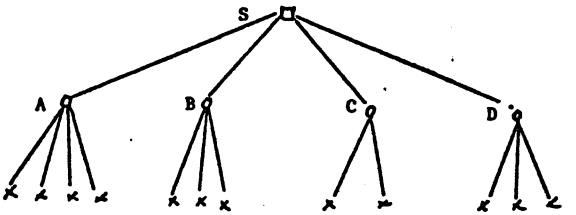
x stored record

● cluster centroid

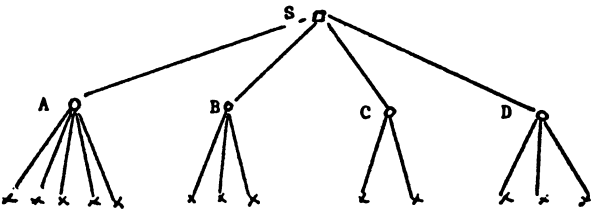
■ centroid of record space

Typical Clustered File Organization

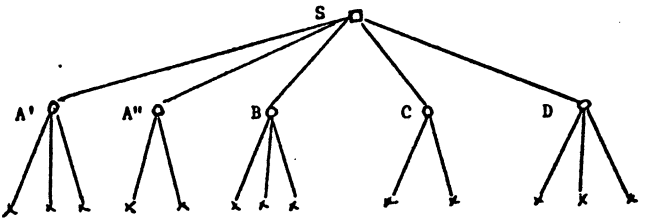
Fig. 2



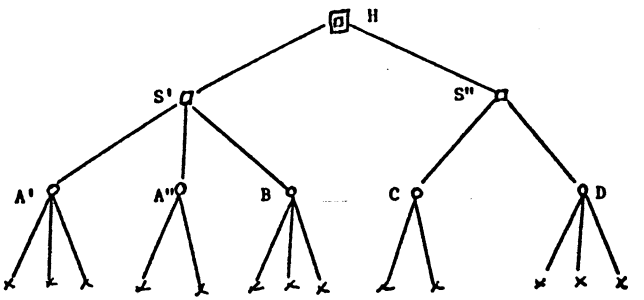
a) Initial State of Cluster Structure



b) Addition of One More Item to Cluster A



c) Splitting Cluster A into Two Pieces A' and A"

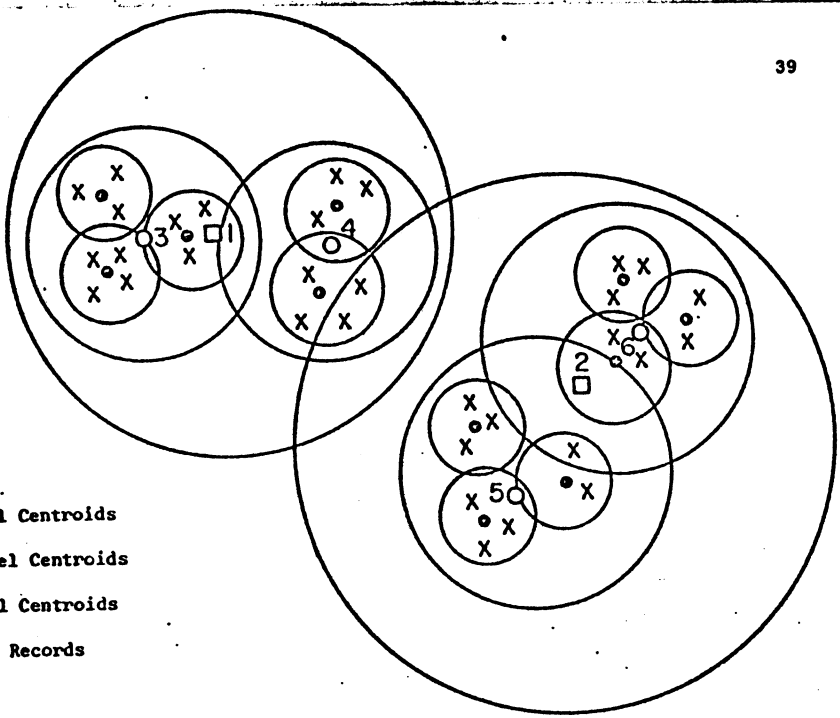


d) Splitting Supercluster S into Two Pieces S' and S"

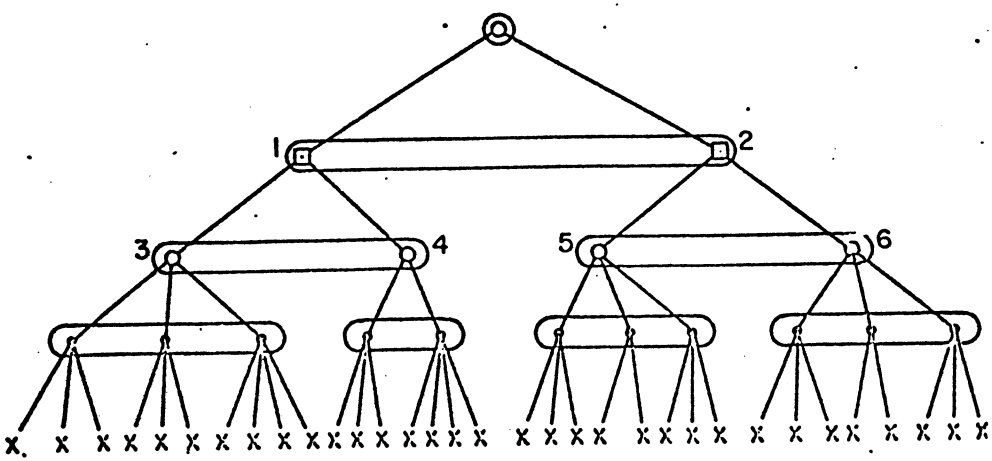
Top level centroid

Second level centroid

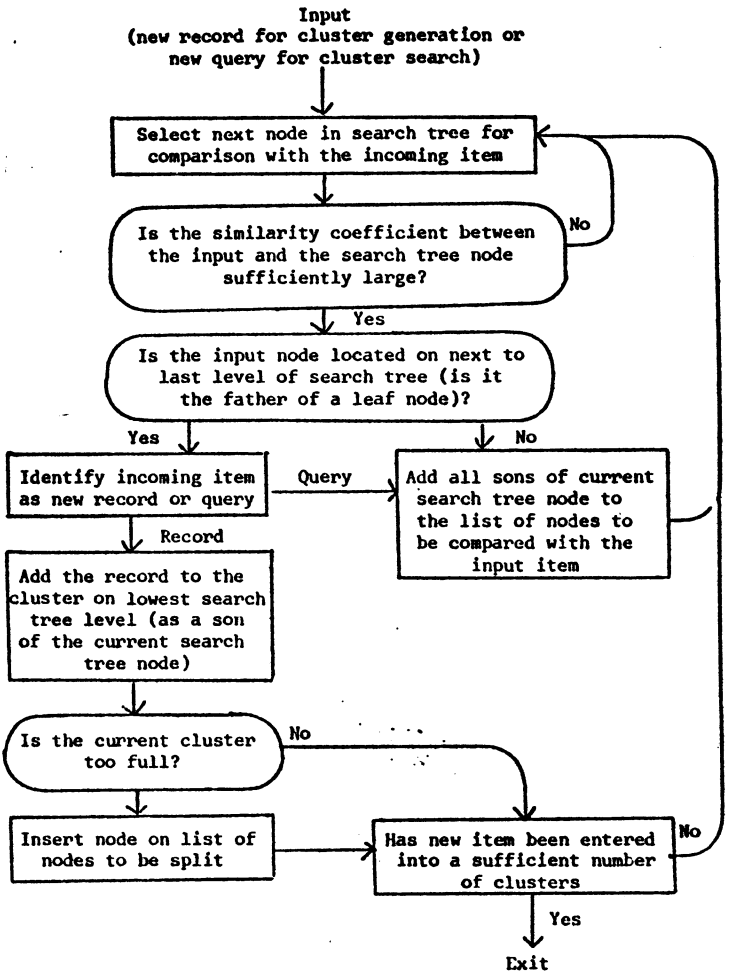
Example of Cluster Splitting Process



a) Clustered File Organization



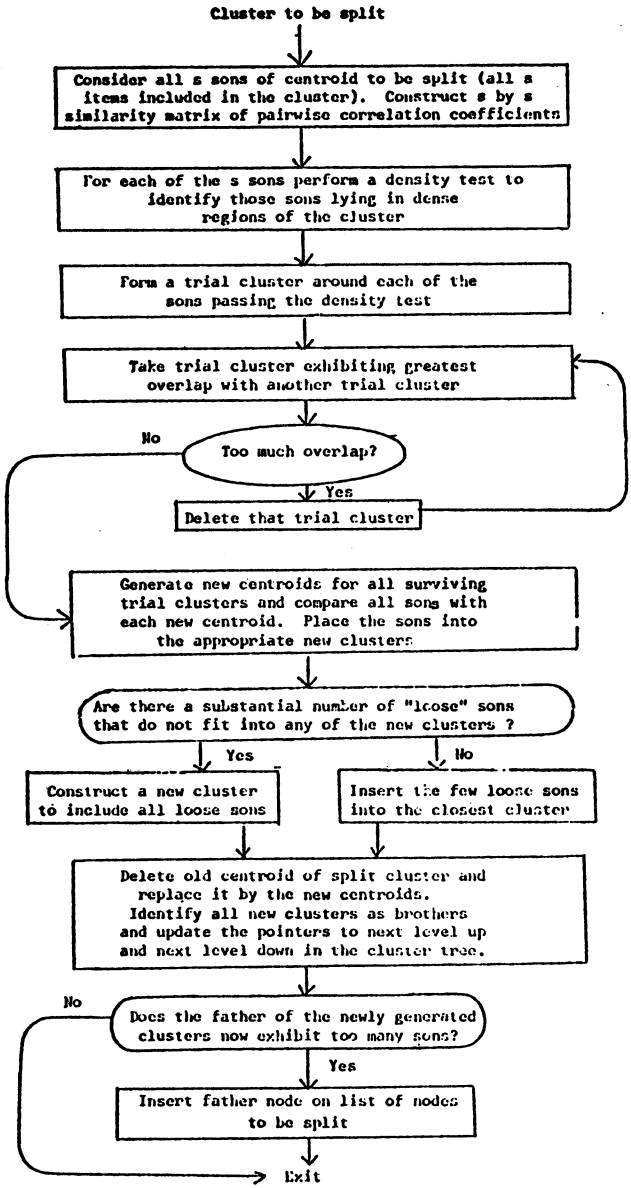
b) Search Strategy for Clustered File



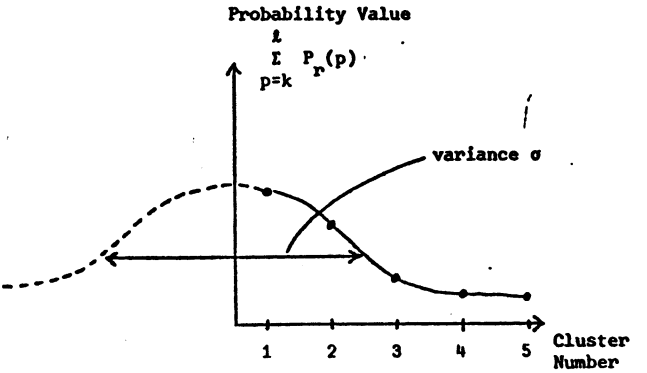
Cluster Generation and Search

(adapted from [12])

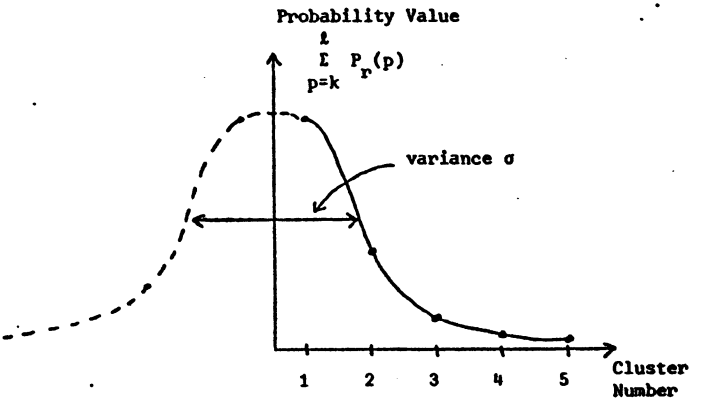
Fig. 5



Cluster Splitting Process
(adapted from [12])



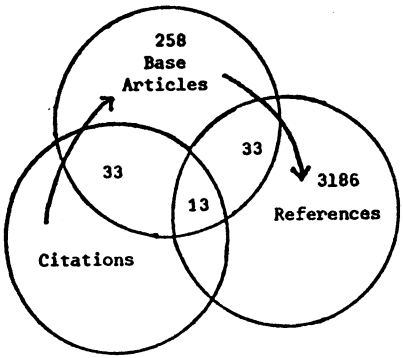
a) Sample Distribution of P-values for Fixed k=α



b) Sample Distribution of P-values for k=2α

Variations of Variance with Changes in Threshold Value k

Fig. 7



334 base articles

3,232 distinct references from base articles

3,520 distinct collection items

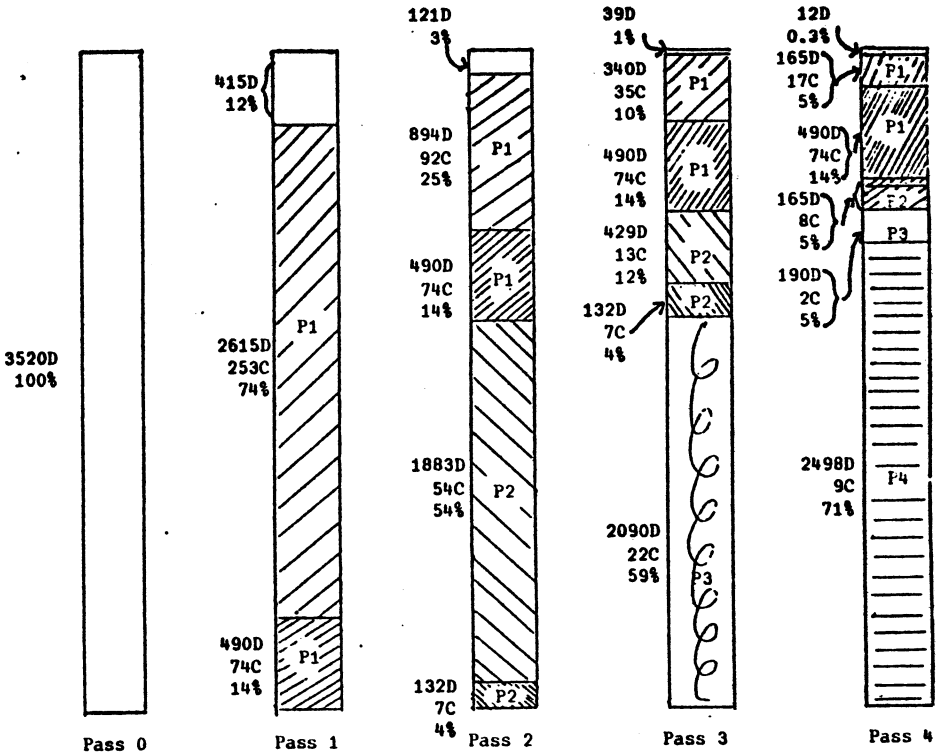
3,893 distinct links

Make-up of Computer Science Collection

(1974 ACM literature plus 1968 and 1971 articles

Computer and Control Abstracts)

Fig. 8



Cluster Generation Process

Fig. 9

D documents
C clusters
% percent of clustered documents

Expanded	Recall	Precision	Queries
1	0.21091	0.23991	24
2	0.35477	0.21720	24
3	0.41550	0.17295	24
4	0.47346	0.14680	24
5	0.52047	0.12769	24
6	0.55952	0.11316	24
7	0.56369	0.09877	24
8	0.58012	0.08840	24
9	0.61566	0.08269	24
10	0.63087	0.07688	24
12	0.69880	0.07113	24
14	0.74603	0.06577	24
16	0.77713	0.05966	24
18	0.79356	0.05472	24
20	0.81605	0.05077	24

a) Cosine Function

Expanded	Recall	Precision	Queries
1	0.21377	0.24694	24
2	0.29110	0.18579	24
3	0.33650	0.14723	24
4	0.38534	0.12389	24
5	0.41012	0.10611	24
6	0.45195	0.09540	24
7	0.50342	0.08972	23
8	0.51592	0.08155	22
9	0.54355	0.07834	22
10	0.57654	0.07540	22
12	0.62157	0.06943	22
14	0.63300	0.06230	22
16	0.66332	0.05834	19
18	0.70850	0.05624	19
20	0.73386	0.05395	18

b) Probability Function (Equation (1))
Fixed k = 2

No. of Clusters Expanded	Recall	Precision	No. of Queries
1	0.26750	0.28576	20
2	0.36927	0.27391	18
3	0.42314	0.25134	16
4	0.44281	0.23396	16
5	0.47338	0.22599	16
6	0.48148	0.21451	15
7	0.50989	0.21176	14
8	0.53708	0.20879	14
9	0.55592	0.20587	12
10	0.55592	0.20220	12
12	0.57268	0.19732	12
14	0.57615	0.19336	10
16	0.60888	0.19245	10
18	0.61235	0.19038	9
20	0.61698	0.18872	9

c) Probability Function (Equation (1))
Fixed k = 4

No. of Clusters Expanded	Recall	Precision	No. of Queries
1	0.25474	0.29456	24
2	0.29943	0.18805	24
3	0.35664	0.15230	24
4	0.38210	0.12287	24
5	0.41035	0.10824	24
6	0.46260	0.09777	24
7	0.48907	0.08761	24
8	0.53537	0.08518	23
9	0.56763	0.08034	23
10	0.60061	0.07757	23
12	0.63268	0.06955	23
14	0.64411	0.06271	23
16	0.67906	0.05850	21
18	0.72076	0.05587	20
20	0.74682	0.05357	19

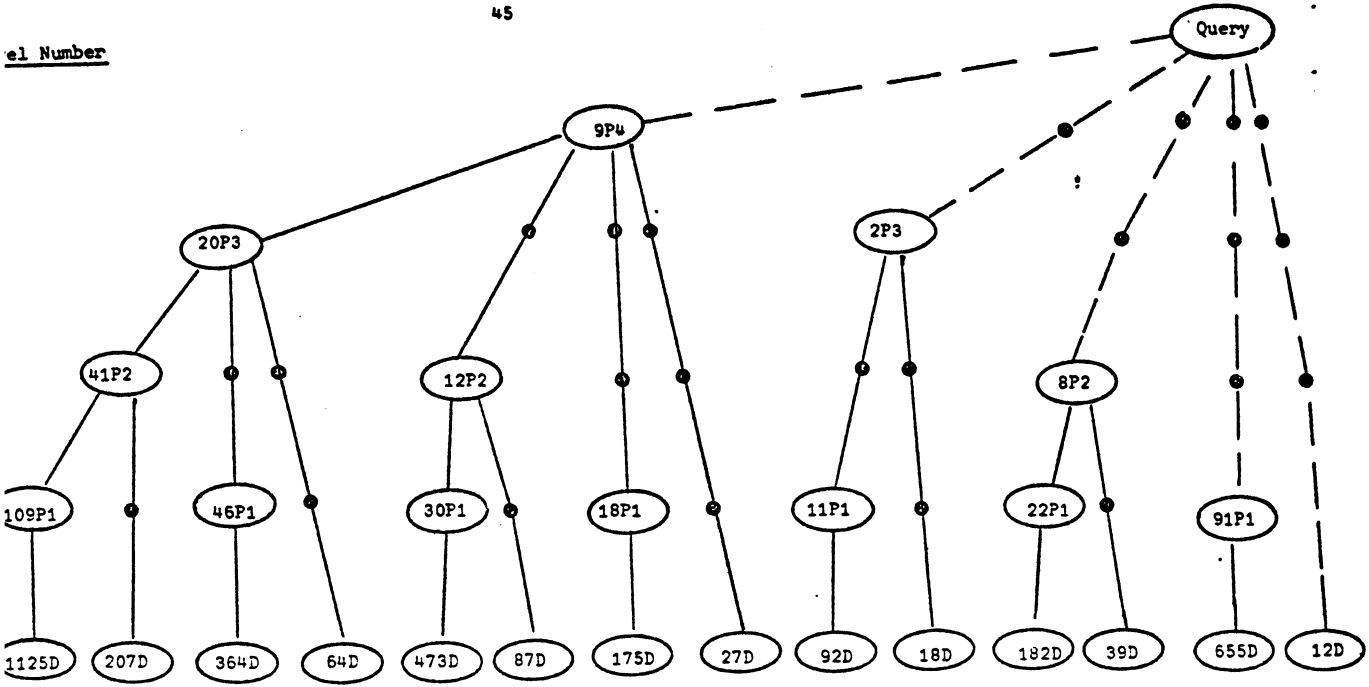
d) Probability Function (Equation (1))
Varying k

Variations of Recall-Precision Values with Number of Expanded Clusters
(averages for 424 documents and 24 queries)
Table 2

Average Number of Clusters Expanded (for 24 queries)	Maximum 7 Clusters Used		Maximum 20 Clusters Used	
	Stop on <1 Relevant 7-1	Stop on <3 Relevant 7-3	Stop on <1 Relevant 20-1	Stop on <3 Relevant 20-3
Fixed k=1	7.0	6.8	20.0	17.0
Fixed k=2	5.45	4.95	14.0	9.1
Fixed k=3	4.5	3.1	9.3	4.5
Fixed k=4	1.96	1.93	2.1	2.1
Varying k	5.5	4.87	14.0	7.2
Variance	3.75	2.7	7.6	4.4

Average Number of Clusters Expanded
for Varying Search Strategies
Table 3

el Number



document

pass-i cluster

empty nodes

Unbalanced Cluster Search Tree

Fig. 10

k	$E_r(k)$	Number of Documents Nonrelevant	Relevant
1	7	0	0
2	(6.9)	1	2
3	(4.9)	0	2
4	-	0	1
5	-	0	1

a) Cluster 38 (6 relevant out of 7)
Query 1 (12 terms)

k	$E_r(k)$	Number of Documents Nonrelevant	Relevant
1	7.8	5	0
2	2.9	3	0
3	(0.4)	0	0
4	-	0	0
5	-	0	0

b) Cluster 50 (0 relevant out of 9)
Query 1 (12 terms)

k	$E_r(k)$	Number of Documents Nonrelevant	Relevant
1	5.9	4	0
2	2.5	1	0
3	(0.5)	1	0
4	-	0	0
5	-	0	0

c) Cluster 38 (0 relevant
Query 2 (5 terms)

k	$E_r(k)$	Number of Documents Nonrelevant	Relevant
1	6.4	0	0
2	4.3	1	0
3	(1.8)	0	2
4	-	0	0
5	-	1	0

d) Cluster 68 (2 relevant out of 7)
Query 2 (5 terms)

Expected Number of Records Having at Least k Properties in Common with Q

Table 1

Recall	Precision	Precision	Precision
0.05	0.30468	0.29463	0.31513
0.10	0.30468	0.29463	0.31513
0.15	0.29873	0.26880	0.29084
0.20	0.29595	0.26255	0.28489
0.25	0.28837	0.25954	0.26489
0.30	0.23844	0.25306	0.26142
0.35	0.23150	0.22509	0.21205
0.40	0.22227	0.22509	0.21205
0.45	0.26392	0.18627	0.17091
0.50	0.19400	0.18329	0.16793
0.55	0.15340	0.16135	0.14979
0.60	0.12380	0.14833	0.13542
0.65	0.10065	0.10893	0.11012
0.70	0.08874	0.10893	0.10893
0.75	0.08874	0.10893	0.10893
0.80	0.06436	0.04540	0.04540
0.85	0.06436	0.04488	0.04488
0.90	0.06436	0.04486	0.04488
0.95	0.05745	0.03571	0.03571
1.00	0.05745	0.03571	0.03571
	Cosine 3 clusters	Fixed k=3 7-3(3.1)	Variance 7-1(3.75)

a) Expanding Approximately 3 Clusters

Recall	Precision	Precision	Precision	Precision
0.05	0.31331	0.30066	0.31195	0.31855
0.10	0.31331	0.30066	0.31195	0.31855
0.15	0.30582	0.27941	0.28915	0.30274
0.20	0.30506	0.27223	0.28320	0.29282
0.25	0.29759	0.25944	0.26320	0.28003
0.30	0.25637	0.25640	0.25279	0.27975
0.35	0.25126	0.21025	0.20925	0.22835
0.40	0.23665	0.21025	0.20925	0.22835
0.45	0.22602	0.18214	0.15527	0.18721
0.50	0.22304	0.17917	0.14535	0.18423
0.55	0.19168	0.16808	0.14110	0.17601
0.60	0.15381	0.15834	0.13537	0.16560
0.65	0.13066	0.12735	0.11855	0.12609
0.70	0.11467	0.12474	0.11226	0.12229
0.75	0.11467	0.12474	0.11226	0.12229
0.80	0.07675	0.05928	0.04340	0.05632
0.85	0.06436	0.03571	0.04488	0.04488
0.90	0.06436	0.03571	0.04488	0.04488
0.95	0.05745	0.03571	0.04481	0.03571
1.00	0.05745	0.03571	0.04481	0.03571
	Cosine 5 clusters	Fixed k=2 7-1(5.45)	Variance 20-3(4.4)	Varying k 7-1(5.5)

b) Expanding Approximately 5 Clusters

Recall-Precision Output for Cosine and Probability Matching Functions

Recall	Precision	Precision	Precision	Precision
0.05	0.31373	0.20294	0.32646	0.32049
0.10	0.31373	0.21248	0.32646	0.32049
0.15	0.30817	0.20736	0.30217	0.30626
0.20	0.30805	0.20017	0.29785	0.29634
0.25	0.30345	0.19730	0.27785	0.28197
0.30	0.26334	0.19447	0.27438	0.28281
0.35	0.25739	0.15837	0.23084	0.23725
0.40	0.24340	0.15837	0.23084	0.23725
0.45	0.23276	0.13026	0.19014	0.19369
0.50	0.22978	0.13026	0.19071	0.19071
0.55	0.19168	0.11256	0.18145	0.18145
0.60	0.15390	0.08677	0.17009	0.17009
0.65	0.13724	0.07359	0.13911	0.13911
0.70	0.12125	0.06246	0.11992	0.11992
0.75	0.12125	0.06246	0.11471	0.11471
0.80	0.09319	0.05678	0.04540	0.04540
0.85	0.08080	0.05184	0.04488	0.04488
0.90	0.07432	0.05184	0.04488	0.04488
0.95	0.05745	0.05184	0.04481	0.04481
1.00	0.05745	0.05184	0.04481	0.04481
	Cosine	Fixed k=1	Variance	Varying k
	7 clusters	7-1(7.0)	20-1(7.6)	20-3(7.2)

c) Expanding Approximately 7 Clusters

Recall	Precision	Precision	Precision
0.05	0.30633	0.30066	0.32162
0.10	0.31228	0.30066	0.32162
0.15	0.31278	0.28203	0.30843
0.20	0.31266	0.27486	0.30013
0.25	0.30806	0.26471	0.28998
0.30	0.29386	0.26419	0.28978
0.35	0.26790	0.22163	0.24466
0.40	0.26741	0.22163	0.24466
0.45	0.24953	0.19619	0.20392
0.50	0.24655	0.19322	0.20095
0.55	0.21959	0.18727	0.19271
0.60	0.20083	0.17577	0.18077
0.65	0.16059	0.15141	0.14604
0.70	0.13281	0.14081	0.13664
0.75	0.12978	0.13964	0.13547
0.80	0.10329	0.06822	0.06189
0.85	0.09759	0.05543	0.05416
0.90	0.09608	0.05543	0.05416
0.95	0.08678	0.05543	0.05409
1.00	0.08678	0.05543	0.05409
	Cosine	Fixed k=2	Varying k
	14 clusters	20-1(14.0)	20-1(14.0)

d) Expanding Approximately 14 Clusters

Recall-Precision Output for Cosine and Probability Matching Functions

Pass Number	Number of Original Documents (loose)	Number of Existing Clusters	Number of Documents per Pass i Cluster	Average Number of Documents for all Existing Clusters	Number of Remaining Distinct Items
0	3520	0	0	0	3520
1	415	327	9.5	9.5	742
2	121	227	33.0	15.45	348
3	39	151	95.0	23.05	190
4	12	110	274.3	31.89	122

Clustering Statistics

Table 5

Encyclopedia			Computing Reviews			Cosers		
Major Topics	Sub Categories	Research Clusters	Major Topics	Sub Categories	Research Clusters	Major Topics	Sub Categories	Research Clusters
Basic Terminology	2	0						
Professional and Educational Aspects	6	0	General Topics and Education	7	0			
Management Societal Legal Economic Aspects	11	0						
History	3	0	Computing Milieu	6	2			
Hardware	5	2	Hardware	6	3	Hardware	9	3
			Analog Computers	6	0			
Software	7	8	Software	8	8	Programming Languages	8	6
						Programming Methods	4	3
Computer Systems	7	6				Operating Systems	5	6
						Data Management	15	7
Applications	27	11	Applications	9	9	Applications	9	0
			Functions	5	2	Artificial Intelligence	6	7
						Symbol Processing	3	2
Theory	9	5				Theory of Computation	7	8
Mathematics for Computer Science	7	21	Mathematics of Computation	9	33	Numerical Computation	18	14
		87			47			57

Major Topic Classes for Three Computer Science Taxonomies

Table 6

Good research