Experiments on Incremental Clustering

Fazli Can
Miami University, commons-admin@lib.muohio.edu

MIAMI UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE
& SYSTEMS ANALYSIS

TECHNICAL REPORT: MU-SEAS-CSA-1991-002

Experiments on Incremental Clustering

Fazli Can

EXPERIMENTS ON INCREMENTAL CLUSTERING

by

Fazli Can
Systems Analysis Department
Miami University
Oxford, Ohio 45056

Working Paper #91-002 08/91

EXPERIMENTS ON INCREMENTAL CLUSTERING

Fazli CAN
Department of Systems Analysis
Miami University
Oxford, OH 45056

Abstract

Clustering of very large document databases is essential to reduce the space/time complexity of information retrieval. The periodic updating of clusters is required due to the dynamic nature of databases. An algorithm for incremental clustering at discrete times is introduced. Its complexity and cost analysis and an investigation of the expected behavior of the algorithm are provided. Through empirical testing, it is shown that the algorithm is achieving its purpose in terms of being cost effective, generating statistically valid clusters that are compatible with those of reclustering, and providing effective information retrieval.

Categories and Subject Descriptors: H.2.2. [**Database Management**]: Physical Design - access methods; H.3.2. [**Information Storage and Retrieval**]: Information Storage - file organization; H.3.3. [**Information Storage and Retrieval**]: Information Search and Retrieval - *clustering, retrieval models, search process*

General Terms: Algorithms, Design, Performance

Additional Key Words and Phrases: cover coefficient, information retrieval, information retrieval effectiveness, dynamic information retrieval environment, cluster validity

1. INTRODUCTION

An *information retrieval system* (IRS) locates and retrieves documents that are relevant to user queries. In the literature IR is also known as *text* or *document retrieval.* In general, IR is done by using *document representatives*. Two common approaches for document representative generation are document *signatures* and *vector space model* [23]. The former uses a bit map array for each document whose entries are set by a hash function using the words of documents as its input [12]. In the vector space model, which is also the approach used in this study, each document is represented by a document vector describing the words, or terms, which appear in the associated document. A document database then simply becomes a matrix. We call this the D, document, matrix. For a database of m documents defined by n terms, an element in the D matrix in row i (document i) at column j (term j), d_{ij} ($1 \leq i \leq m$, $1 \leq j \leq n$), represents the importance, for example number of occurrences, of term j (t_j) to document i (d_i). Sometimes the matrix merely contains 1's and 0's, a binary matrix, to indicate the presence or the absence of terms in documents. Storing the D matrix requires much less space than storing all documents in their entirety. More importantly, it makes the information retrieval easier. The D matrix can be generated manually or by automatic indexing. The set of terms $\{t_1, t_2, \ldots t_n\}$, V, used for the description of documents is called the *indexing vocabulary.*

After storing the document representatives, a method for searching and retrieving must be provided. One of them is the *full search* (FS) method using *inverted indexes* [26]. In this method, associated with each term of V there is a list of <document no., weight> pairs for each document in which that term appears. Regardless of how the actual *query* (user's information request) is produced it is possible to generate a query vector. All that is required then is to traverse only query term lists using a *matching function* , looking for documents containing those terms. The matching function is used to decide which documents are potentially relevant (i.e., match the query) and should be returned to the user [21, 24]. It is up to the user to determine which of those retrieved documents are actually relevant. Various retrieval techniques use this generic approach to retrieve documents. An overview of retrieval techniques and citations for the related literature can be found in [2].

Like any other information system, two concepts, *efficiency* and *effectiveness,* are the main concerns of an IRS. Efficiency and effectiveness are, respectively, associated with the time it takes the system to perform the search and with the quality of retrieval. In IR, clustering is introduced to improve both efficiency and effectiveness of retrieval. Within the context of IR, a *cluster* is a homogeneous group of documents that are more strongly associated with each other than with those in other groups. Forming these groups is called *clustering* [1]. The reason behind clustering is that documents that are strongly associated tend to be relevant to the same query (which is known as "clustering hypothesis" in the IR literature) [23, 24]. A typical implementation of *cluster-based retrieval* (CBR) first matches queries with cluster representatives, called *centroids*, then matches the queries with documents in the selected clusters to find those that actually match the query [7, 26].

The importance of CBR comes from browsing capability, i.e., the facility to see the documents which are similar to the relevant documents [23, p. 345]. During browsing efficient access to document information can be achieved by storing the documents of a cluster in closer proximity in the disk environment [19, pp. 309-314]. The clustering concept is also valuable in application of hypertext technology to IR. Hyperlinks between nodes (documents) can automatically be generated using clusters.

Document databases are dynamic in nature. In 1986 it was estimated that about 100 million unique documents were searchable online with an update rate of 10 million unique documents during the year. From 1976 to 1986 there was an eight-fold increase in the number of documents available online [14, p.45]. In a document database the change in the composition of documents also changes V, since new documents may introduce new terms which do not exist in the old documents and deleted documents may be the only users of some terms. This implies that an IRS must be congenial with this dynamic environment. In this study our concern is update of clustered files in such environments.

For cluster file update there are basically three approaches.

(1) *Reclustering*: creating clustering structure from the very beginning.

(2) *Instant maintenance*: maintaining clustering structure at the time of the addition/deletion of documents.

(3) *Incremental (delayed) maintenance*: modifying the given clustering structure at discrete times.

The first approach is inefficient. The second approach is desirable if documents are added/deleted one by one and if the immediate updating is necessary; therefore, it is appropriate for environments such as office automation systems, "accident and incident" systems used by police to build up files of current reports as they become available. The examination of commercial document databases shows that the third approach, incremental maintenance, is suitable for document databases since they are updated periodically. (The file size and the update characteristics of some DIALOG databases are provided in Table I.)

Table I. File Size and Update Characteristics of Some Commercial Document Databases [10]

Database	File Size	Updates
Computer Database	330,091	Every two weeks
Confer. Papers Index	1,309,711	Six times per year
ERIC	660,868	Monthly
McGraw-Hill News	79,640	Every 15 minutes
Medline	6,200,000	Approx 25,000/mon.

In this paper we introduce a cluster maintenance algorithm. Since we have a series of consecutive updates at discrete times it is referred to as incremental clustering. The algorithm of this study draws its roots from the Cover-Coefficient-based Clustering Methodology (C^3M) [7] and, therefore, is called Cover-Coefficient-based Incremental Clustering Methodology (C^2ICM).

Different from our previous research [3, 4] on cluster maintenance, this work studies multiple maintenance steps with dynamic indexing vocabulary. Furthermore, the algorithm is based on a new data structure yielding very low computational complexity with negligible storage overhead. This makes C^2ICM suitable for very large dynamic document databases. In this study we also examine the effect of database dynamics on maintained clustering structure and introduce a new Monte Carlo approach to test and validate the similarity of different clustering structures for a given database. The validation of the obtained clustering structures is done by another set of Monte Carlo experiments. The IR performance assessment of the algorithm is also provided by using the most promising matching functions available in the IR literature [22].

The structure of the paper is as follows. In Section 2 a brief review of the clustering and cluster maintenance problem is provided. In Section 3 the base concept of the incremental clustering algorithm, the cover-coefficient (CC) concept, and its relation to clustering are described for the reader who is not familiar with it. Section 4 introduces the incremental clustering methodology, C^2ICM, its complexity and cost analysis, and analytical analysis of its expected behavior in dynamic document database environments. Section 5 covers our experimental design and evaluation. In this section it is shown through empirical testing that C^2ICM is achieving its purpose, i.e., generating statistically valid clusters that are compatible with those of reclustering while attaining lower cost than that of reclustering. The retrieval experiments showed that IR effectiveness of C^2ICM is as effective as C^3M which is known to have superior performance with respect to other algorithms of the literature [7]. The experiments have been carried out using two document databases: the TODS322 database of 322 documents and 58 queries, and the INSPEC database of 12684 documents and 77 queries. The conclusion is given in Section 6. The appendix provides an example application of the algorithm C^2ICM to clarify the concepts and the data structures used in the development of the algorithm.

2. CLUSTERING AND CLUSTER MAINTENANCE

Clustering algorithms can be classified into various groups using various criteria. One such criterion is clustering methodology. According to this criterion, we have three basic categories: *Graph theoretical*, *single pass*, and *iterative* algorithms. A typical graph theoretical algorithm prepares a similarity matrix representing the similarity between individual documents then applies a similarity threshold to determine the clusters represented by closely related documents. Each cluster forms a connected graph. Depending on connectivity of documents, the structures known as "single link," "group average," and "complete linkage" are formed.

An example for the single-pass algorithm is the *seed oriented* clustering. In this approach, a cluster initiator (e.g., a document), called *cluster seed*, is determined for each cluster to be formed. Documents are then assigned to the clusters of the seeds to which they are most similar. For implementation the number of clusters must be known. This is usually provided as an input parameter. The *nearest-neighbor* (NN) algorithm uses each document as a seed; each cluster contains the "seed" and the document most similar to the seed, i.e., its nearest neighbor. Note that this approach may end up having the same number of clusters as the number of documents.

Iterative algorithms try to optimize a clustering structure according to an optimization function. The generation of the initial clusters can be done, for example, by using seed oriented clustering.

Clustering algorithms can also be classified according to the manner in which documents are distributed to clusters. Some classifications are *partitioning*: where clusters cannot have common members; *overlapping*: where cluster can have common members, and *hierarchical*: a tree structured organization of clusters and documents where leaves of the tree represent documents and upper levels of the tree represent clusters of clusters or documents (or both); the root being the super cluster containing all of the document database.

The Cluster Maintenance Problem: A Critical Overview

The problem of cluster maintenance deals with the modification of clustering structures due to addition of new documents or deletion of old documents (or both). The addition and deletion of documents, respectively, may also imply addition and deletion of terms. That is, the environment is dynamic both in terms of documents and indexing vocabulary, V, used for database description.

A close examination of clustering algorithms reveals that most of them are not suitable for clusters maintenance [24, p.58]. In the literature there are very few maintenance algorithms. A brief critical overview of cluster maintenance approaches is provided in this section. In general, these algorithms are developed for growing databases; however, most of them can also be used in case of deleted documents.

The algorithm of Crouch has three processes [8]. The *categorization process*, generates groups of terms (*category vectors*) which represent a subdivision of the document database. In the *clustering process*, the category vectors are used like cluster seeds and the documents are assigned to the "categories" they have resemblance with. Later, during the *update process*, the category vectors that have common terms with the new documents are modified or new category vectors are generated. The new and modified category vectors are then used as queries and compared with the documents. The matching documents of the "queries" and the new documents are assigned to their respective categories (i.e., clusters). The major problems of this algorithm are the following. (1) The generated clustering structure depends on the order of processing of the documents and particularly on the initial set of documents. (2) Each document is processed at least twice (once for categorization and once for clustering). (3) New documents may modify many of the previous category vectors, and as a result, the maintenance can be as expensive as reclustering.

Another approach for maintenance, *cluster splitting*, treats a new document as a query and compares it with the existing clusters and assigns it to the cluster that yields the best match [20]. The major problems of this algorithm are the following [19, p. 494]. (1) The generated clustering structure depends on the order of processing of the documents. (2) The generated clustering structure starts to degenerate with relatively small increases (such as 25 percent to 50 percent), i.e., frequent reclustering may be needed in fast growing database environments. (3) The clustering structure obtained can be lopsided (few fat and many thin clusters) which in turn results in inefficient searches.

Adaptive clustering is based on user queries. The method introduced in [29] assigns a random position on the real line (the hypothetical line from negative infinity to positive infinity) to each document. For each query, the positions of relevant *k* documents are shifted closer to each other. To ensure that all documents are not bunched up together the *k* documents are selected randomly and pulled away from their centroids. This approach has some disadvantages: (1) The clusters generated by such an approach depend on the order of query processing; (2) The definition of clusters (separation between the members of different clusters) requires an input parameter and the appropriate value for this parameter is database dependent; and (3) The speed of convergence can be slow. The applicability of such algorithms in dynamic environment is not yet known.

For the maintenance of clusters generated by graph theoretical methods (single link, group average, complete link) similarity values need to be used. First, similarity calculations among the m_2 number of new documents imply a complexity of $O(m_2^2)$. Then the similarity among the existing m_1 documents and the new documents needs to be calculated yielding an overall complexity of $O(m_1 \times m_2 + m_2^2)$ which can be approximated as $O(m_1 \times m_2)$ since $m_1 >> m_2$. For the single link case, the similarity values among old documents are not needed. The group average and the complete linkage methods require the complete knowledge of similarities among old documents [25, pp. 29-30]. This means a running time of $O(m_1^2)$ or a storage requirement of the same order. This is followed by the clustering process, which is an additional cost. That is, the time and space requirements of the group average and the complete link approaches are prohibitive. Furthermore, they have an inherent problem: They do not necessarily define a unique clustering structure for a given set of similarity values among individual documents of a database [25, p. 27]. The efficient implementation of complete-linkage method makes it dependent upon the order in which documents in the database are processed [9]. The update cost of the single-link method is reasonable, $O(m_1 \times m_2)$; however, its IR effectiveness is known to be unsatisfactory [11, 25, 27].

The maintenance of "clusters" generated by the NN algorithm is relatively easy and the order of computations is $O(m_1 \times m_2)$ [13]. In addition to this computational requirement, for each old document its nearest neighbor and the similarity value between them are needed. This is a storage requirement in the order of m_1 which can be regarded as reasonable. However, it should be noticed that the structure due to the NN algorithm is hardly a clustering structure since the average cluster size is two. That is, maintenance of NN structure is an easy update of a "so called" clustering structure.

Another possibility for handling database growth is using a cheap clustering algorithm of *mlogm* complexity to recluster the entire database consisting of old and new documents. However, even this approach would be prohibitive, especially when considering consecutive maintenance. Furthermore, the assumptions which make the complexity of the cheap algorithm *mlogm* may not be valid for cluster maintenance. Another point is the following: even though some algorithms have theoretical complexity of *mlogm* the experimental evaluations of these have shown that due to large proportionality constant the actual time taken for clustering is greater than that of for an m^2 algorithm [24, p. 59]. These points imply that we would be better off by using an efficient maintenance algorithm instead of reclustering the documents.

3. PRELIMINARY CONCEPTS

In this section we describe the preliminary concepts of the C^2ICM algorithm very briefly. The details can be found in [7]. The base of the algorithm, the CC concept, provides a measure of similarities among documents. The concept is first used to determine the number of clusters and cluster seeds, then to assign nonseed documents to the clusters initiated by the seed documents.

The CC concept determines document relationships in a multidimensional term space by a two-stage probability experiment. The experiment randomly selects terms from documents in two stages: the first stage randomly chooses a term t_k from document d_i, then the second stage tries to choose the selected term t_k from d_j. For d_i the above experiment is repeated for all terms and for all documents and we obtain c_{ij} ($1 \leq i, j \leq m$), the probability of success. Intuitively this is a measure of similarity, since the documents that have many common terms will have a high probability of being selected because they appear together in most term lists. The experiment is also affected by the distribution of terms in the whole database: If d_i and d_j share rare terms this increases c_{ij}.

Formally, for a D matrix of size m by n, c_{ij} is defined as follows.

$$c_{ij} = \alpha_i \times \sum_{k=1}^{n} (d_{ik} \times \beta_k \times d_{jk}) \qquad \text{where } 1 \leq i, j \leq m$$

where α_i and β_k are the reciprocals of the ith row sum and kth column sum. To apply the above formula each document must have at least one term and each term must appear at least in one document.

Based on this, we construct a mxm C (*cover-coefficient*) matrix of c_{ij} values for a given D matrix. Some characteristics of the C matrix are the following [7].

(1) $0 < c_{ii} \leq 1$, $0 \leq c_{ij} < 1$;

(2) $c_{ii} \geq c_{ij}$ and $\min(c_{ii}) = 1 / m$ for a binary D matrix;

(3) $(c_{i1} + c_{i2} + \ldots + c_{im}) = 1$;

(4) $c_{ij} = 0 \longleftrightarrow c_{ji} = 0$, $c_{ij} > 0 \longleftrightarrow c_{ji} > 0$, and in general $c_{ij} \neq c_{ji}$;

(5) $c_{ii} = c_{jj} = c_{ij} = c_{ji} \longleftrightarrow d_i$ and d_j are identical;

(6) d_i is distinct $\longleftrightarrow c_{ii} = 1$.

Thereby from (4) it is seen that c_{ij} measures an asymmetric similarity between d_i and d_j.

In the C matrix, a document having terms in common with several documents will have c_{ii} value considerably less than 1. Conversely, if a document has terms in common with very few documents, then its c_{ii} value will be close to 1. The c_{ij} entry indicates the extent to which d_i is "covered" by d_j. (The reason for the phrase choice "cover-coefficient" is given in [7].) Identical documents are covered equally by all other documents; if the document vector of d_i is a subset of the document vector of d_j, then d_i will cover itself equally as well as d_j covers d_i. It is also true that if documents have no terms in common, then they will not cover each other at all, and the corresponding c_{ij} and c_{ji} values will be zero.

Because higher values of c_{ii} result from document d_i having terms found in very few of other documents, we let $c_{ii} = \delta_i$ and call δ_i the *decoupling coefficient* of d_i. It is a measure of how different d_i is from all other documents. Conversely we call $\psi_i = 1 - \delta_i$ the *coupling coefficient* of d_i.

Similar to documents, the relationship between terms can be defined via the C' matrix of size n x n whose entries are defined as follows.

$$c'_{ij} = \beta_i \times \sum_{k=1}^{m} (d_{ki} \times \alpha_k \times d_{kj}) \qquad \text{where } 1 \leq i, j \leq n$$

Using c'_{ij} the same concepts of decoupling, coupling, and number of clusters (see below) can be defined for the terms.

In a database with similar documents, the diagonal entries (δ_is) of the C matrix will be low; however, if the database contains dissimilar documents, the diagonal entries will be high. So the number of clusters is defined as follows.

$$n_c = \sum_{i=1}^{m} \delta_i \qquad \text{where } (1 \leq n_c \leq \min(m, n))$$

It is shown that for a given D matrix, the number of clusters indicated by documents (i.e., by δ_is) and terms (i.e., by δ'_js) is the same and then $\max(n_c) = \min(m, n)$. Hence, the average document cluster size, d_c, is in the range of max(1, m/n) to m. We can see that n_c in CC is directly related to (a function of) the D matrix, and is not just some arbitrary value or an input parameter. This provides flexibility and adds robustness to the methodology.

In forming the initial clusters we employ the C^3M. Our incremental clustering algorithm, C^2ICM, is an extension of C^3M. In C^3M clusters "grow" from seed documents. For the selection of seed documents we compute the cluster seed power of each document. The cluster *seed power*, p_i, of d_i is defined as follows

$$p_i = \delta_i \times \psi_i \times \sum_{j=1}^{n} d_{ij} \qquad \text{and} \qquad p_i = \delta_i \times \psi_i \times \sum_{j=1}^{n} (d_{ij} \times \delta'_j \times \psi'_j).$$

corresponding to the binary and weighted version of the D matrix, respectively. In the second formula $\delta'_j = c'_{jj}$ and $\psi'_j = (1 - \delta'_j)$. Here, δ_i provides separation of clusters through document dissimilarities and ψ_i provides intra-cluster cohesion through document similarities. The third term, the summation, provides normalization. Thereafter the documents corresponding to the documents with the n_c highest seed powers are selected as the clusters seeds. In a document database it is possible to have identical or almost identical documents; only one of the "identical" documents can be used as a seed. If the seed powers of two documents significantly differ (using a threshold) then this means that the documents are distinct. This test works almost one hundred percent of the time. For further details of the false seed elimination process refer to [7]. For an example see the Appendix.

After determining n_c seed documents, for all remaining $(m - n_c)$ nonseed documents we determine the cluster seed which maximally covers them. If there is more than one cluster seed that meets this condition, the nonseed document is assigned to the cluster whose seed power is the greatest among the candidates. That is, each document can be a member of only one cluster. To calculate the extent to which a nonseed document, d_i, is covered by a seed document, d_j, implies the calculation of c_{ij}. This computation involves summing many zero terms, since the D matrix is very sparse. The elimination of those zero entries is done by considering only nonzero d_{ik} and d_{jk} entries. This is accomplished by traversing a term list for each term of d_i. A term list contains <document seed number, term weight> pairs for each seed in which that term appears. This data structure is referred to as Inverted term Index for Seed Documents (IISD). Using IISD we calculate the extent to which d_i is covered by seed documents not seed by seed, but together for all seed documents in an incremental manner. During the traversal of the list, c_{ij} values for the corresponding seed documents are updated. This approach is like the implementation of FS using inverted term indexes [7, 26]. The appendix provides an example to help understanding the concepts of the algorithm.

4. THE INCREMENTAL CLUSTERING ALGORITHM

In this section we first introduce the C²ICM algorithm, then its complexity and cost analysis. An analysis of the effects of the database dynamism on the generated clustering structures is also included.

4.1 The C²ICM Algorithm

Incremental clustering starts with m_1 documents. These documents are clustered using the C³M algorithm. After this, clusters are updated due to newcomers (additions) and obsolete documents (deletions) using the C²ICM algorithm. In the rest of the paper the symbols m' and m'', respectively, indicate the number of added and number of deleted documents; similarly $D_{m'}$ and $D_{m''}$, respectively, indicate the set of added and deleted documents. D_m indicates the current document database.

A brief description of C²ICM is given in the following;the symbols *"U"* and "-", respectively, indicate the set operations union and difference.

C²ICM :

[a] *Compute the cluster seed powers of the documents in the updated document database, $D_m = D_m \cup D_{m'} - D_{m''}$ and pick the cluster seeds. (In general $m' >> m''$.)*

[b] *Determine D_r, the set of documents to be clustered. Cluster these documents assigning them to the cluster of the seed that covers them most.*

[c] *If there were documents not covered by any seed, then group those together into a rag-bag cluster.*

[d] *Apply the above steps for each document database update.*

The set D_r consists of the newcomers, the members of the ragbag cluster of the previous step, and the members of the falsified old clusters. An old cluster is defined to be false if

(1) Its seed is not a seed anymore (deleted seeds would have their clusters falsified also);

(2) One or more of its nonseed documents becomes a seed after a database update.

The reader should refer to the appendix for an example application of the algorithm.

The C²ICM algorithm is order independent (i.e., the clustering structure is independent of the order in which documents are processed) for the members of D_r only. Order independence is a desirable feature for a clustering algorithm. This somewhat less desirable order-dependence situation is offset by the cost savings (see Section 4.3) and good IR performance of the algorithm.

4.2. Complexity Analysis of the Algorithm

In this section the complexity analysis is given for a weighted D matrix since it represents the more general case. Like C³M, the implementation of C²ICM does not require the construction of the complete C matrix: the data structure IISD is used to obtain the necessary c_{ij} values. It should be stated that the storage overhead of the IISD is very low. For example, for the INSPEC database the whole data structure contains 23049 entries with a storage overhead of just 5.6 percent of the D matrix. (The D matrix of the INSPEC database contains 412255 nonzero entries, see Table III.)

To cluster the initial set of m_1 documents (before increment) we use C^3M. Its computational requirement is as follows [7].

$$3 \times x_d \times m_1 + m_1 \times \log m_1 + m_1 \times x_d \times t_{gs}$$

where x_d and t_{gs}, respectively, indicate average number of distinct terms per document and average number of seeds per term. In the above formula, ($3 \times x_d \times m_1$) indicates the order of the computations needed to calculate the number of clusters (δ_i values for all documents) and seed powers (which also requires δ'_i values for all terms) of individual documents. The second term, ($m_1 \times \log m_1$), is needed for seed selection (sorting). The third term, ($m_1 \times x_d \times t_{gs}$), is for clustering, i.e., assigning nonseed documents to clusters initiated by seed documents. Actually the number of documents to be clustered is ($m_1 - n_c$), since $m_1 >> n_c$ ($m_1 - n_c$) is taken as m_1.

The cost of generating the IISD is ignored since it is very low, ($n_c \times x_d$). The term ($m_1 \times \log m_1$) can be ignored with respect to the others. In general, for very large databases we expect to observe $3 << t_{gs} << t_g$, where t_g is the average number of documents per term. Accordingly the complexity of C^3M is ($m_1 \times x_d \times t_{gs}$). This complexity is considerably less than those of the most other clustering algorithms whose complexity range is $O(m^2)$ or $O(m^3)$ [25, 27].

During incremental clustering we can assume equal sized additions. To make analysis simpler we ignore document deletions. However, this does not invalidate the analysis since $m' >> m''$ in a typical environment.

The computational cost of the first incremental step is the following.

$$3 \times x_d \times (m_1 + m') + (m_1 + m') \times \log (m_1 + m') + r \times m' \times x_d \times t_{gs}$$

where r is the ratio of documents to be clustered to m'. In the experiments we observed that the maximum r is equal to two ($r \geq 1$). In the above expression the first term determines the complexity since it is larger than the other two terms. For example, for a very large database, such as Medline, it is expected that the following equality would be true ($m_1 / m' >> r \times t_{gs}$). Accordingly, the complexity of one incremental clustering step is $O(x_d \times (m_1 + m'))$.

Now let us consider the complexity of incremental clustering for k increments. We can assume that x_d is almost the same during increments. Notice that it is very hard, if not impossible, to determine the optimum level of the depth of indexing for a given database [17] However, we may assume that the average depth of indexing, x_d, remains more or less the same throughout different stages of a database. This is because, the documents of the database usually have comparable sizes and we expect to observe similar indexing criteria (automatic or manual) applied to all documents. Hence, x_d would be almost the same for a group of documents. This is a reasonable assumption and is observed in the experiments (see Table III). Then overall complexity for k increments becomes the following.

$$x_d \times [(m_1 + m') + (m_1 + 2 \times m') + \ldots + (m_1 + k \times m')] =$$
$$x_d \times [k \times m_1 + (1 + 2 + \ldots + k) \times m']$$

Accordingly the complexity of the algorithm becomes $O(k \times m_1 \times x_d + k^2 \times m' \times x_d)$. If $(k^2 \times m') >> (k \times m_1) \dashrightarrow (k \times m') >> m_1$. Then the complexity of the algorithm becomes $O(k^2 \times m' \times x_d)$. However, if m_1 is very large with respect to m', then the complexity of the algorithm for k incremental steps becomes $O(k \times m_1 \times x_d)$. For example, for the Medline database (refer to Table I) m_1 = 6,200,000 and m'= 25,000 (m_1 / m'= 248). For $(k \times m') >> m_1$ to hold we need more than twenty years since in Medline updates are done monthly! This may be hard to observe. Therefore for very large databases the complexity of the algorithm for k incremental steps depends on the initial size of the database, and is equal to $O(k \times m_1 \times x_d)$.

Reduction of Complexity

Before continuing let us consider the following question: "Can we reduce the complexity of C^2ICM by saving the previous values of δ_i and p_i for all documents, and δ'_j for all terms?" The answer is "No," since the magnitude of recomputations to update these data structures would be as large the calculation of their values from the very beginning.

To illustrate this let us consider δ_i values. If δ_i for all d_i is saved (call this value $\delta_{i,old}$) after a database increment, $\delta_{i,old}$ must be updated if the terms of d_i also appear in $D_{m'}$ or $D_{m''}$. The new δ_i value, $\delta_{i,new}$, can be calculated as follows:

$$\delta_{i,new} = \delta_{i,old} - \alpha_i \times d_{ij}^2 \times \beta_{j,old} + \alpha_i \times d_{ij}^2 \times \beta_{j,new}$$
$$= \delta_{i,old} + \alpha_i \times d_{ij}^2 \times (\beta_{j,new} - \beta_{j,old})$$

Where $\beta_{j,old}$ and $\beta_{j,new}$, respectively, indicate the β_j value of term j before and after the database increment. If all terms of d_i were used by the database increment, then the cost of the above update would be the same as the calculation of δ_i from the very beginning (100 percent recalculation). Figure 1 shows the usage of the terms for the INSPEC database with the increase of the database size. In this experiment m_1 = m'= 1,000 documents (the last increment contains 684 documents) and the V is allowed to grow with the addition of new documents. That is, new documents of each step use terms of the already existing documents and new terms which have not been used so far. As would be expected, as we increase the size of the database and the V, the percentage of the old terms used by the new documents decreases (shown by P_1 in the figure). However, the reused terms always count more than 90 percent of all term occurrences in old documents (shown by P_2 in the figure) and this incurs the same amount (more than 90 percent) recalculations. This is valid even for very small database increments (shown by P_3). For example, the last increment of the experiment increases the database size by 6 percent (684/12000) and the percentage of the recalculations is 92!

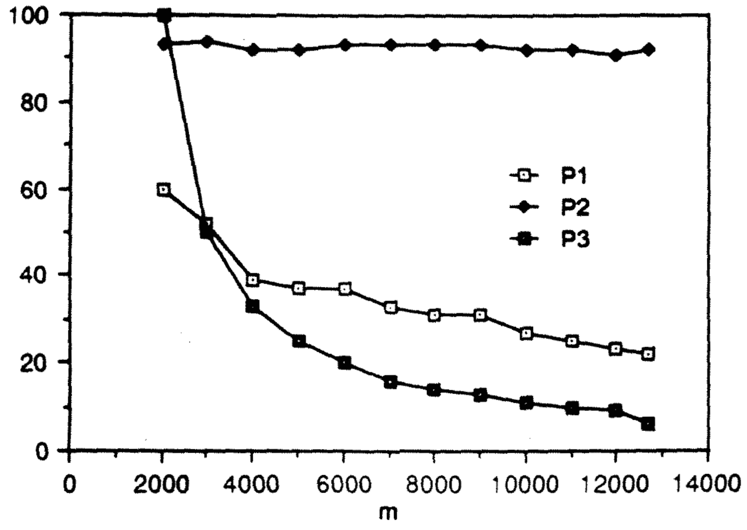

Figure 1. The percentage of the old terms reused in new documents (P_1), the percentage of all term occurrences of the reused terms in the old documents (P_2), and the percentage of the database size increase (P_3) for m_1 =m'= 1,000, INSPEC database.

This discussion shows that the complexity figure $x_d \times (m_1 + m')$ will remain the same (at least) due to recalculations of the δ_i values. Therefore, we cannot reduce the complexity by saving the previous values of the basic data structures of the algorithm. However, for databases with very small increments (e.g. for Medline the size of increase is 0.4%) the saving approach has the potential of validity. In such environments, if we save the previous values of the data structures, the complexity of the algorithm would be $O(r \times m' \times x_d \times t_{gs})$, or more correctly $O(m' \times x_d \times t_{gs})$, since r is usually very small (in our experiments $r \leq 2$).

4.3. Cost Analysis

Our complexity analysis assumes that the proportion of the reclustered documents is very low with respect to the previous size of the database. For the analysis to hold this assumption must be true. In the worst case, i.e., if all seeds are falsified, C^2ICM degenerates to a reclustering process. If the number of documents to be reclustered is zero, this indicates that incremental clustering does not incur any extra cost. That is, it clusters only the newcomers. Therefore, the cost of the incremental clustering algorithm is proportional to the number of reclustered documents, R. (At this point we must state that we do not want to have a "costless" maintenance algorithm, since if the cost is zero we would not be able to reflect the effect of the additions/deletions on the old database. In other words, we want some "reclustering" to happen during incremental handling, but not that much.)

Similarly, the cost of the reclustering algorithm can be taken as the number of reclustered documents. For k incremental steps the number of reclustered documents is equal to

$[(m_1) + (m_1 + m') + \ldots + (m_1 + (k\text{-}1) \times m')] = [k \times m_1 + (k \times (k - 1) / 2) \times m'] =$
$[k \times m_1 + (k \times (k - 1) / 2) \times m']$

Then the proportional cost of incremental clustering with respect to reclustering becomes $2 \times R / [2 \times k \times m_1 + k \times (k - 1) \times m']$ which, together with the observations from our experiments, shows that incremental clustering is much more efficient than reclustering. This is because $2 \times R << [2 \times k \times m_1 + k \times (k - 1) \times m']$. We know that the reclustering algorithm, C^3M, generates valid clustering structures and it provides an effective CBR environment [5, 7]. The analysis of C^2ICM from these points of views is provided in Section 5.

4.4.The Effect of Database Dynamics on the Output of the Algorithm

The CC concept reveals the relationships between indexing and clustering. The analytical derivation and experimental validation of these relationships are provided in [7]. In this section we use these relationships to foresee the effect of *database dynamics* (i.e., the change in m and n) on clustering structure.

The CC-based indexing-clustering relationships are formulated as follows.

$$n_c = t / (x_d \times t_g) = (m \times n) / t = m / t_g = n / x_d, \text{ and } d_c = m / n_c = t_g$$

The meaning of m, n, x_d, t_g, n_c, and d_c is as before, and t indicates the total number of nonzero entries in the D matrix. As explained before, we assume that the average depth of indexing, x_d, is the same throughout different stages of a database. Notice that individual documents may show considerable divergence from the average and our assumption does not ~~assume~~ anything for the variance.

Number of Clusters

Let $n_{c,k-1}$, $n_{c,k}$ and n_{k-1}, n_k, respectively, indicate the number of clusters and number of terms at k-1'st and k'th database increments (k= 0 indicates the initial database). By using the relationship $n_c = n / x_d$ the entity $n_{c,k}$ can be rewritten as follows.

$$n_{c,k} / n_{c,k-1} = (n_k / x_d) / (n_{k-1} / x_d) = n_k / n_{k-1} \quad \text{or}$$
$$n_{c,k} = (n_k / n_{k-1}) \times n_{c,k-1} \qquad (\text{for } k > 0)$$

Accordingly, the estimated number of clusters at step (increment) k, $n_{c,k}$, is a function of the number of clusters in the previous step ($n_{c,k-1}$), the previous size of the V, n_{k-1}, and the current size of the V, n_k. The formula also indicates that as we increase the size of V, n_c increases. Normally we expect that the database documents would eventually include all possible terms, hence V and n_c would cease growing. However, in practice, many of the vocabularies appear to grow indefinitely, although their rate of growth decreases with increase in size of the database [15, p. 206]. Therefore, the number of clusters grow indefinitely, but its rate of growth decreases with increase in database size.

Average Cluster Size

The discussion above also indicates a steady increase in average cluster size (d_c), since normally we expect a steady increase in database size and slower rate of increase for n_c. The same is also implied by $d_c = t_g$. Recall that t_g is defined as (t / n) and we expect to observe a higher rate of increase in t with respect to the size of V, n, due to steady increase in database size.

The relationship between d_c and the other variables of the system can also be expressed as follows.

$$d_c = m / n_c = m / (n / x_d) = (m / n) \times x_d$$

The above equation indicates that if x_d remains the same, m and n can be used to estimate the average cluster size, d_c, as follows.

$$d_c \begin{array}{l} > x_d \text{ if } m > n \\ < x_d \text{ if } m < n \end{array}$$

5. THE EXPERIMENTS

Our experiments were designed to

(1) observe the effect of dynamic environment on the clustering structure in terms of number of clusters and average cluster size,

(2) show that C^2ICM is cost effective,

(3) test the similarity between the clusters generated by C^2ICM and those generated by C^3M,

(4) test the validity of the clustering structure generated by C^2ICM in the statistical sense,

(5) test the compatibility of CBR effectiveness of C^2ICM and C^3M using several matching functions to show that the former is as effective as the latter, which is known to be very effective for CBR.

In this section we first describe the databases used and present the experimental results.

5.1. The Document Databases

In this study we use two document databases: TODS322 and INSPEC. The TODS322 database is from the papers published by the Association for Computing Machinery in the journal *Transactions on Database Systems*. The database contains 322 documents from March 1976 through September 1989. Each document contains the title, keywords given by the author(s), and the abstract. The indexing vocabulary is generated from the stems found in the documents of the database. A stop word list is used to avoid the common words of the English language. A smaller version of TODS322 (TODS214) was used in our earlier studies [3, 4, 7]. The details of indexing software can be found in [18]. The second database, INSPEC, contains 12684 documents covering topics in computer science and electrical engineering. The D matrix (and the queries) of the INSPEC database is common with other studies [7, 11, 22, 25, 26]. This database is one of the largest test databases of the IR literature.

5.2. The Clustering Experiments

We created six experiment cases for each database choosing different values for m_1, but always keeping the same size of increments. No deletions were done. For each database we used cases where m_1= 32 percent, then 45 percent, 59 percent, 73 percent and 86 percent of the database. We added documents to the initial m_1 in steps of about 14 percent of the full size of the databases (44 and 1734 documents, respectively, for TODS322 and INSPEC databases). The actual numbers of documents are shown in Table II.

Table II. The Size of the D Matrices Used in the Experiments

Matrix No.	1	2	3	4	5	6
D. B. size (% of full size)	32	45	59	73	86	100
m_1 / TODS	102	146	190	234	278	322
m_1 / INSPEC	4014	5748	7482	9216	10950	12684

For example, if we begin with matrix number 1, this provides us five increments (steps). For INSPEC it grows from 4014 documents with increments of size 1734 and matrix number 6 contains all documents available in the database.

Throughout the experiments we used dynamic indexing, i.e., the documents of each increment are allowed to use new terms which have not been used by the old documents. That is matrices 1 through 5 are subsets of the complete D matrix (matrix number 6). The tests were done both on weighted and binary D matrices. (In a weighted D matrix d_{ij} indicates the number of occurrences of term j in document i.) The results of both cases are similar; however, the weighted version always outperforms the corresponding binary version. In this paper we only report the results of the weighted cases since a weighted D matrix is more general than binary one. The experimental results for the binary version of the TODS322 database are available in [6].

Table III. Characteristics of the D Matrices

	INSPEC				TODS322			
Matrix No	n	t	t_g	x_d	n	t	t_g	x_d
1	8009	128284	16.02	31.96	1406	5273	3.75	51.70
2	9637	183459	19.04	31.92	1689	7488	4.43	51.29
3	11074	242569	21.90	32.42	2008	9864	4.91	51.92
4	12452	301446	24.21	32.71	2242	12223	5.45	52.24
5	13537	356867	26.36	32.59	2414	14585	6.04	52.46
6	14573	412255	28.29	32.50	2602	17011	6.54	52.83

The characteristics of the D matrices of the experiments are shown in Table III. As we defined previously, n is the number of terms; t is the number of nonzero entries in the D matrix, t_g is (t / n), and x_d is (t / m). Table III shows that for a given D matrix, x_d, depth of indexing, remains almost the same throughout the steps of incremental clustering. This was our assumption in the complexity analysis and in the analysis to foresee the output behavior of the algorithm. (As a sidelight we must state that x_d of individual documents show considerable variations. For example, the variance of depth of indexing for all INSPEC documents (matrix number 6) is 203.55. Our assumption is even valid for such an environment.)

If we begin with matrix number 1 there are five incremental steps. The number of clusters for each step (k) is shown in Table IV. (In this table, the row for k= 0 displays the initial conditions just before incremental clustering, which are obtained by C^{3}M using matrix number 1.) The same table also shows that the estimated n_c values for steps k ($1 \leq k \leq 5$), $n_{c,k} = (n_k / n_{k-1}) \times n_{c,k-1}$, are very close to the corresponding actual n_c values. Similarly, the estimated d_c, $(m / n) \times x_d$, values are very close to the corresponding actual d_c values. These results demonstrate that we can estimate the values of n_c and d_c by the main variables of the document database, namely, database size, m, and size of V, n. This information is valuable to predict the future requirements of an IRS in terms of secondary storage size and retrieval time.

Table IV. The Values of n_c, Est. n_c d_c, Est. d_c for Initial Clustering (k = 0) and Incremental Clustering (k > 0)

	INSPEC				TODS322			
Step (k)	n_c	Est. n_c	d_c	Est. d_c	n_c	Est. n_c	d_c	Est. d_c
0	269	*	15	16	25	*	4	4
1	321	324	18	19	30	30	5	4
2	364	369	21	22	36	36	5	5
3	407	409	23	24	39	40	6	6
4	442	443	25	26	43	42	7	6
5	475	476	27	28	46	46	7	7

(*) Est. (n_c) can be calculated for k > 0

The incremental clustering behavior of the algorithm for the INSPEC database is given in Table V. In this table the abbreviations NFC and NFD, respectively, indicate the number of falsified clusters and the number of falsified documents. The table shows that with $m_1 = 4014$, the first incremental step (43 percent increase in database size, $m' / m_1 = 1734 / 4014 = 0.43$) falsifies 22 percent (60 out of 269) of the old clusters containing 989 documents. The next step falsifies 72 clusters and the number of falsified documents is 1460. For $m_1 = 5748$ the first increment falsifies the same number of clusters as the second increment of $m_1 = 4014$. This is because, both steps of the algorithm use the same previous D matrix with 5748 documents and the same set of documents as the increment. Accordingly, they both falsify the same number of clusters, actually the same seeds, but not necessarily the clusters containing exactly the same documents. This can be seen from the total number of falsified documents, for m_1 = 5748 the first increment falsifies 1381 documents, for m_1 = 4014 the second increment falsifies 1460 documents. The same discussion is also valid for the rest of the table. For example, in the last row of Table V, for all cases of m_1, the number of falsified clusters is 52; however, the number of falsified documents are not the same but close to each other (all observations are within the range of 1204 to 1289). This is an indication of the close similarity of the falsified clusters.

Table V. Incremental Clustering Behavior of the Algorithm for the INSPEC Database

	m_1									
	4014		5748		7482		9216		10950	
m	NFC	NFD	NFC	NFD	NFC	NFD	NFC	NFD	NFC	NFD
5748	60	989	-	-	-	-	-	-	-	-
7482	72	1460	72	1381	-	-	-	-	-	-
9216	76	1550	76	1534	76	1543	-	-	-	-
10950	48	992	48	1008	48	1062	48	1084	-	-
12684	52	1281	52	1289	52	1257	52	1204	52	1248

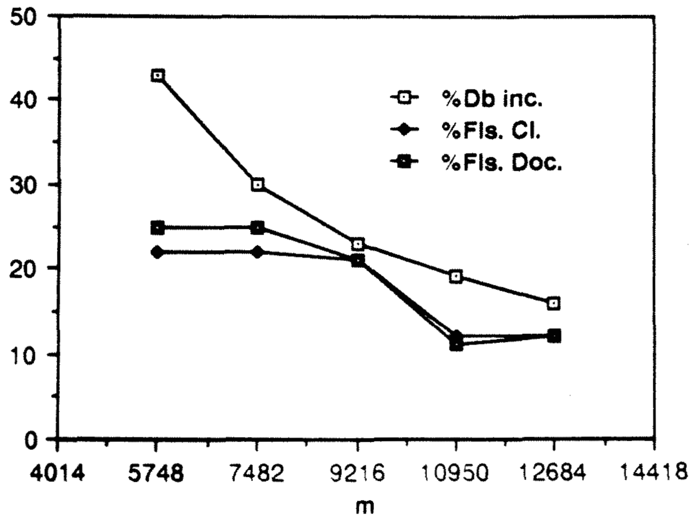

Figure 2. The behavior of the algorithm for the INSPEC database (m_1 = 4014, m' = 1734, for five increments).

For the INSPEC database the case m_1 = 4014 provides the maximum number of incremental steps in the experiments and reflects the behavior of the algorithm for other values of m_1; therefore, it deserves more attention. With m_1 = 4014 we have five incremental steps and the total number of falsified documents for all steps is 6272 (= 989 + 1460 + 1550 + 992 + 1281) documents. Without incremental clustering, each step requires reclustering of all old documents available at that step and we have to recluster 37410 (= 4014 + 5748 + 7482 + 9216 + 10950) documents. Therefore, the cost of incremental maintenance is just 17% (6272 / 37410) of reclustering, i.e., it is cost effective. Figure 2 shows that as we increase the size of the database (or decrease the proportional size of the increments) the percentage of the falsified clusters and the falsified documents decrease and the cost effectiveness of the algorithm increases. This is an expected behavior, since smaller increments should not considerably affect an already existing clustering structure. The experimental results for the TODS322 database are similar and can be seen in [6].

5.3. The Similarity Experiments

The purpose of the similarity experiments is to check how well the maintenance approach is achieving its purpose, i.e., generating clusters that are compatible with those of reclustering while attaining lower cost than that of reclustering. For this purpose we measure the similarity between the clusters generated by incremental clustering (C^2ICM) and the clusters generated by reclustering (C^3M). These measures are then compared against the case where the incremental clusters are filled with documents randomly. Monte Carlo experiments are performed to obtain the distribution of the similarity measures when the documents are randomly distributed in the clusters generated by C^2ICM . In the experiments, 1000 random cases were produced. This helps to show that the clusters generated by C^2ICM are significantly effective in placing the documents of clusters generated by C^3M into fewer clusters than that of the random case. This establishes that the clusters generated by incremental clustering and their corresponding similarity to the clusters of reclustering did not happen by chance in a statistical sense.

To perform similarity tests, three different types of similarity measure were used. Similar results were obtained from all of them. The first measure is *Corrected Rand* (CR) [16]. It has a maximum value of one when the clusters (partitioning structures) are identical, and a value of zero when the clusters are generated by chance, i.e., CR is corrected for chance. The second measure is the *Goodman-Kruskal* (GK) metric [16] which is similar to the chi-square based association measures. It has a maximum value of one for identical clusters, and a minimum value of zero. The third similarity measure is an *intuitive metric* (IM) which we introduce in this study. In IM, for each cluster, C, of C^3M we find the number of C^2ICM clusters, x, which contain at least one member of C. For a given C, x can assume a value between 1 and $\min(n_c, |C|)$, where $|C|$ indicates the size of C. In the experiments $n_c >> d_c$ (average cluster size) for both databases; therefore, the IM results are unbiased by the values of n_c. After considering all clusters of C^3M, an average value is obtained for x. The average has a value of one for identical clustering structures of C^3M and C^2ICM. For GK and IM there is no correction for chance.

In this paper we report our findings for CR and IM statistics. The results of GK are in agreement with the observations of CR; however, observed similarity values are higher than those of CR since the GK measure does not have any correction for chance. The results of the CR similarity measures for both databases are shown in Figure 3. The matrix number 1 indicates the case when $m_1 = 102$, $m_1 = 4014$ for TODS322 and INSPEC, respectively. For this case we have five incremental steps to reach a full database containing all documents. For both databases, this means 316 percent increase in the collection size. The case with matrix number 2 provides us four incremental steps. (Refer to Table II for matrix sizes of the other cases.) As expected, increase in m_1 (or decrease in the number of incremental steps) implies higher similarity.

The experimental results in terms of the number of clusters to be opened (IM) are given in Table VI. In this table, the average values for Monte Carlo (random) experiments are given in row R1000 (Random 1000). The average values for the Monte Carlo experiments were also obtained for theoretical "perfect random" using a modified version [7] of Yao's theorem [28] for estimating number of block accesses. These values are shown in the row PR (Perfect Random) of Table VI. The PR and R1000 values are almost identical for the TODS322 database. We can also say the same for the INSPEC database. This is an indication that the average of 1000 random cases is sufficient to give us a good sense of "perfect random." The closer values of PR and R1000 for the TODS322 case (with respect to the INSPEC case) can be explained by the smaller size of the database. Since TODS322 is smaller we are more successful in sensing perfect randomness using 1000 random experiments. In this table, the last row (A) indicates the "actual" IM observations using the output of C^2ICM. The A values are always better (lower) than that of the random case.

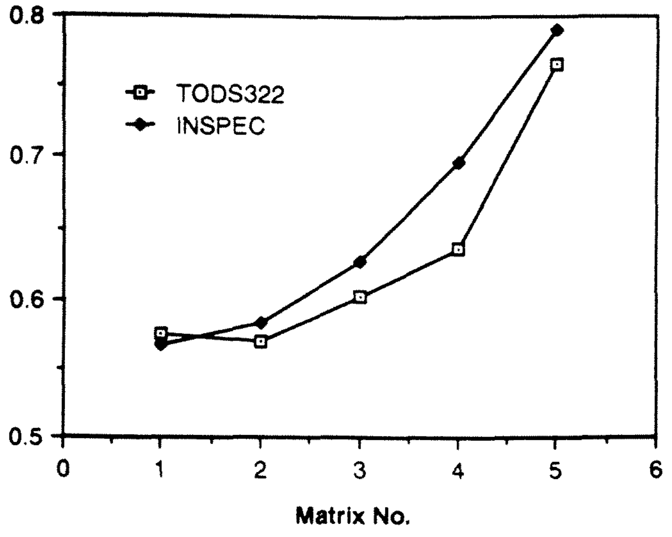

Figure 3. Similarity between C^2ICM and C^3M using CR measure.

Table VI. Similarity Between Incremental Clustering and Reclustering Using IM

D.base	INSPEC					TODS322				
Mtx.No.	1	2	3	4	5	1	2	3	4	5
R1000	25.069	25.057	25.053	25.114	25.195	5.983	5.987	5.945	5.964	6.039
PR	24.430	24.419	24.415	24.471	24.553	5.989	5.993	5.951	5.972	6.048
A	6.758	6.491	5.952	5.082	3.779	2.478	2.500	2.326	2.196	1.891

For final decision on similarity we must show that the observed similarity values (IM, CR, GK) are significantly better than the corresponding random similarity values. (Notice that the random similarity values are not randomly generated, but that the documents are assigned "randomly" to the clusters of the clustering structure generated by C^2ICM.) For the similarity measure IM, and for a given D matrix, the observed A must be significantly lower than the corresponding random similarity values whose average is indicated by R1000. For the similarity measures CR and GK the observed values must be significantly greater than the corresponding random cases. For this purpose we constructed a histogram of the individual random values of GK and IM similarity measures. This is done for all D matrices. No histogram construction is needed for CR. Since all of the random similarity values in terms of CR measure are almost zero (recall that CR is corrected for chance); and therefore, they are significantly different from the observed values (see Figure 3).

All of the histograms are very similar, only one is reported here for the INSPEC database. Additional experimental results for the TODS322 database (including binary indexing) can be found in [6]. The 1000 random IM values, A(r), for matrix number 1 of the INSPEC database were grouped into ten bins, and Figure 5 shows the percentage counts for each bin. That is, the histogram shows the approximate baseline distribution (probability density function) of the A(r) values. The plot shows that the A value of 6.758 is significantly different from the random case, since all of the random observations have a value greater than A. This is significant evidence that the incremental clustering methodology provides clusters that are similar to the clusters generated by reclustering and that they do not happen by chance in a statistical sense. The observations made using weighted indexing are slightly better than those of binary indexing [6].

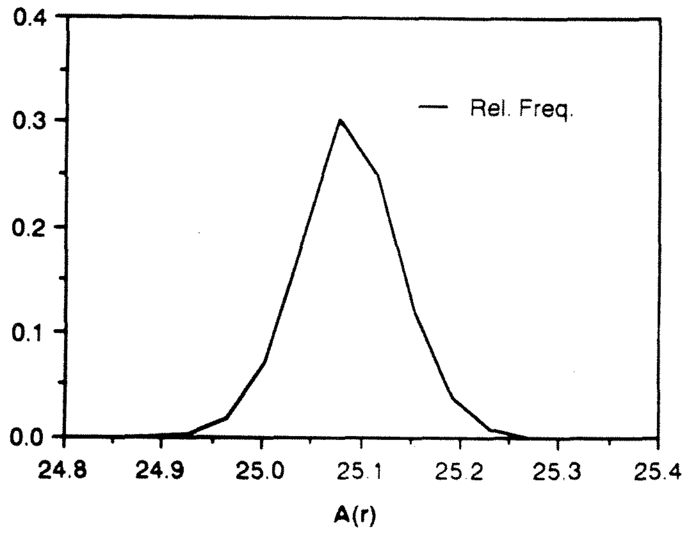

Figure 5. Histogram of the relative frequency of A(r) values for Matrix No. 1 of INSPEC (min= 24.869, max= 25.272, avg.= 25.069, std. dev.= 0.230, bin length= 0.038).

5.4. Validation of Clustering Structures

Before using a clustering structure for any purpose (in our case IR) one must show the clustering structure is good representative of the intrinsic character of the data set being clustered. In other words the clustering structure must be significantly "different" from (better than) random clustering. Such a structure is called valid. The two other cluster validity issues, i.e., clustering tendency and validity of individual clusters are beyond the scope of this study. An in-depth study and an overview of the cluster validity problem from the viewpoint of IR are, respectively, provided in [16] and [5, 27].

The cluster validation methodology of this study is based on the users' judgement on the relevance of documents to queries. Given a query, let a *target cluster* be defined as a cluster which contains at least one relevant document for the query. Let n_t indicate the average number of target clusters for a set of queries based on a given clustering structure. Random clustering is obtained by preserving the same clustering structure and assigning documents randomly to the clusters as we have done in the similarity experiments. Let n_{tr} indicate the average number of target clusters under random clustering. The number of target clusters, for a given query, in the case of random clustering is obtained by using the modified version of Yao's theorem that we use in our similarity experiments in calculating the similarity measure IM.

The case $n_t \geq n_{tr}$ suggests that the tested clustering structure is invalid, since it is unsuccessful in placing the documents relevant to the same query into a fewer number of clusters than that of the average random case. The case, $n_t < n_{tr}$, is an indication of the validity of the clustering structure; however, to decide validity one must show that n_t is significantly less than n_{tr}.

Table VII. Comparison of C^2ICM and Random Clustering in Terms of Average Number of Target Clusters for All Queries (INSPEC Database)

Matrix No.	1	2	3	4	5
n_t	23.558	23.597	23.639	23.831	24.143
n_{tr}	30.436	30.418	30.413	30.506	30.630

Table VII gives the n_t and n_{tr} values for all incremental clustering experiments for the INSPEC database. The characteristics of the queries for both of the databases are given in Section 5.5.3. According to our validity criterion, we must know the probability density function of n_{tr}. For this purpose we generated 1000 random structures for each clustering structure produced by C^2ICM and obtained the average number of target clusters for each random case, $n_t(r)$. Later, these $n_t(r)$ values were used to construct a histogram of $n_t(r)$ values (i.e., approximate probability density function). For example, for matrix number 1 of the INSPEC database, the minimum and maximum $n_t(r)$ values, respectively, are 29.623 and 30.792 (standard deviation is 0.175). This shows that the n_t value of 23.558 for this experiment is significantly different from the random case, since all of the $n_t(r)$ observations have a value greater than n_t. The same is observed for all of the incremental clustering experiments including TODS322. This shows that the clusters are not an artifact of the C^2ICM algorithm; on the contrary, they are valid.

5.5. Information Retrieval Experiments

In this section we assess the effectiveness of C^2ICM by comparing its CBR performance with that of C^3M. In [7] it is shown that C^3M is 15.1 to 63.5 (with an average of 47.5) percent better than four other clustering algorithms in CBR using the INSPEC database. These methods are single link, complete linkage, average link, and Ward method [11]. It is also shown that CBR performance of C^3M is compatible with a very demanding (in terms of storage and CPU time) implementation of a complete linkage method [26]. (This algorithm of the complete linkage method is order independent and also very expensive to use in real IR environment. For example El-Hamdouchi and Willett were unable to use the algorithm for the INSPEC database in the IBM 3083 mainframe environment [11].) In this section we want to show that the CBR behavior of C^2ICM is as effective as C^3M and therefore better than other clustering methods.

In the experiments, m_1 is 4014 and 102 documents, respectively, for INSPEC and TODS322 databases. As we indicated previously this provided us five incremental steps and 316 percent growth in database size with respect to m_1 (see Table II). This is a considerable enlargement. The clustering structure obtained after the fifth increment is used in the retrieval experiments. Notice that these clustering structures are the outcome of the most stringent conditions of our experimental environment.

5.5.1. Evaluation Approach

In CBR, clusters are first ranked according to their centroids' similarity with the user query. Then, n_s number of clusters are selected. Then, d_s documents of the selected clusters are chosen according to their similarity with the query. The selected clusters and documents must have nonzero similarity with the query. Typically a user evaluates the first ten to twenty documents returned by the system and after that he(she) is either satisfied or the query is reformulated. In this study d_s values ten and twenty are used for both databases.

The effectiveness measures used in this study are the average precision for all queries, and total number of queries with no relevant documents, Q. *Precision* is defined as the ratio of the number of retrieved relevant documents to the number of retrieved documents. Another evaluation measure, which is used in the current studies [7, 11, 13, 26] is the total number of relevant documents retrieved for all queries, T. In this work T is not used. This is because, the effectiveness measures, precision and T, result in relatively identical effectiveness.

The relationship between precision and T can be shown as follows. Let r_i indicate the number of relevant documents for query i after examining d_s number of documents. Assuming that for each query the system is able to find d_s documents with nonzero similarity, then for nq queries the average precision can be expressed as

$$[1 / (nq \times d_s)] \times (r_1 + r_2 + \ldots + r_{nq}) = [1 / (nq \times d_s)] \times T$$

where T is equal to $(r_1 + r_2 + \ldots + r_{nq})$, i.e., the total number of relevant documents retrieved for all queries. Therefore, average precision for nq queries is nothing but T divided by $(nq \times d_s)$. The interested readers can easily obtain the T values of our experiments from the given precision values.

5.5.2. Query Matching

For query matching there are several matching functions depending on term weighting components of document and query terms. Term weighting basically has three components: The term frequency component (TFC), the collection frequency component (CFC), and the normalization component (NC) [22]. The weights of the terms of a document and a query (denoted by w_{dj} and w_{qj}, $1 \leq j \leq n$) are obtained by multiplying the respective weights of these three weighting components. After obtaining the term weights, the matching function for a document and a query is defined as the following vector product.

$$\text{similarity}(D, Q) = \sum_{k=1}^{n} w_{dk} \times w_{qk}$$

Salton and Buckley [22] obtained 1800 different combinations of document/query term-weight assignments, of which 287 were found to be distinct. In the same study, six of these combinations are recommended due to their superior IR performance. In this study, we used these six matching functions in additional to the cosine matching function. Each of these matching functions enables us to test the performance of C^2ICM under a different condition.

Due to limited space the definition of matching functions is skipped. In this study, the cosine similarity function is referred to as TW1 (term weighting 1) and the other six are referred to as TW2 through TW7. In [7] they are again referred to as TW1 through TW7; in [22] they are, respectively, defined as (txc.txx), (tfc.nfx), (tfc.tfx), (tfc.bfx), (nfc.nfx), (nfc.tfx), and (nfc.bfx). In a given experiment the same matching function is used both for cluster and document selection.

5.5.3. Retrieval Environment: Queries and Centroids

The query characteristics for both databases are presented in Table VIII. The query vectors are generated in the same manner as the D matrices are created. The query vectors of TODS322 and INSPEC databases are binary and weighted, respectively. In the INSPEC case, the query weight information is used whenever needed. The queries of the INSPEC database were collected at Cornell University and Syracuse University.

The cluster centroids are constructed by using the terms with the highest total number of occurrences within the documents of a clusters. The maximum length (i.e., number of distinct terms) of centroids is a parameter. The maximum centroid lengths for TODS322 and INSPEC databases, respectively, are set as 150 and 250. These values are just 5.77 and 1.72 percent of the total number of distinct terms used for the description of the respective databases. In our previous experiments we showed that after some point the increase in centroid length does not increase the effectiveness of IR and the centroid lengths just mentioned are suitable for these databases [7]. Similar results are also obtained for hierarchical clustering of various document databases [26].

The characteristics of the centroids produced for C^3M and C^2ICM are very similar to each other. The characteristics of the centroids for both databases and algorithms are provided in Table IX. In this table, x_c indicates the average number of distinct terms per "centroid," %n indicates the percentage of terms which appear in at least one centroid, t_{gc} indicates the average number of centroids per term for the terms which appear in at least one centroid, $\%x_c$ is 100 x (x_c / average number of distinct terms per "cluster"), and %D indicates the total size of the centroid vectors as a percentage of t in the corresponding D matrix. The last entry (%D) indicates that the storage cost of the centroids is about one third of that of the D matrix, which is good in terms of search and storage efficiency. Actually, if we had used all the terms of each cluster, the value of %D would have been around 50. For INSPEC with C^2ICM: the average number of distinct terms per cluster is 450.24 hence %D is 52 (n_c x 450.24 / t, where n_c= 475, t= 412255). However, as stated earlier, we know that longer centroids do not result in superior retrieval effectiveness. For example, the retrieval experiments with the maximum centroid length of 500 gives almost the same results as the experiments reported in this study.

Table VIII. Characteristics of the Queries

Database	No. of Queries	Avg. No. of Terms per Query	Avg. No. of Rel. Docs. per Query	Total No. of Relevant Documents	No. of Distinct Docs. Retrieved	No. of Dist. Terms for Query Def.
TODS322	58	25.31	5.26	305	126	572
INSPEC	77	15.82	33.03	2543	1940	577

Table IX. Characteristics of the Centroids (the last row is taken from [7])

Database	Algorithm	x_c	%n	t_{gc}	%x_c	%D
TODS322	C^2ICM	132.70	63	3.72	56	36
	C^3M	138.96	65	3.76	56	38
INSPEC	C^2ICM	227.73	58	12.74	51	26
	C^3M	237.09	60	12.96	51	27

5.5.4. Retrieval Effectiveness

In CBR the first step is to determine the number of clusters, n_s to be selected. The increase in effectiveness would be expected to go up to a certain n_s. After this "saturation" point the retrieval effectiveness remains the same [7]. In our experiments the saturation point for TODS322 and INSPEC are observed for n_s equal to twelve and fifty, respectively. In other words, for TODS322 (n_c = 46) and INSPEC (n_c = 475) 26 percent and 11 percent of the clusters are selected.

For the INSPEC database, for all queries and for all matching functions TW1 through TW7, the average percentage of the matched documents is 12.5 percent (1588 documents) and 14.5 percent (1849 documents), respectively, for C^3M and C^2ICM. The same values for the TODS322 case are 34 percent (110 documents) and 35 percent (114 documents). The number of selected documents slightly varies depending on the matching function, but all of them are very close to the average. For INSPEC the minimum and maximum are observed using TW1 and TW6, respectively. For C^2ICM these values are 1706 and 1897 documents and for C^3M they are 1458 and 1639 documents. These observations indicate that CBR behavior of the algorithms are similar. The comparability of the percentage of selected clusters and percentage of selected documents also indicate that documents are evenly distributed among clusters. The skewed distribution of documents among clusters, i.e., many large and small clusters, is a classical problem of clustering [19, p. 496].

The results of the IR experiments are shown in Table X and XI. The first and second rows of each effectiveness measure (precision, Q) are for d_s values of 10 and 20, respectively. In the case of TODS322, due to the binary nature of query vectors, the matching functions (TW2, TW3) and (TW5, TW6), respectively, are reduced to TW4 and TW7 [7, 22].

Table X. Effectiveness of the Reclustering (R) and Incremental Clustering (I) for the INSPEC Database (the figures for R are taken from [7])

Effective.	TW1		TW2		TW3		TW4		TW5		TW6		TW7	
Measure	R	I	R	I	R	I	R	I	R	I	R	I	R	I
Precision	.287	.277	.418	.419	.396	.401	.401	.397	.388	.383	.382	.379	.352	.343
	.230	.226	.336	.329	.323	.314	.329	.320	.310	.310	.313	.311	.288	.283
Q	17	17	6	6	8	7	6	6	5	4	9	8	7	6
	13	14	3	2	3	4	3	3	4	2	4	3	4	2

Table XI. Effectiveness of Reclustering (R) and Incremental Clustering (I) for the TODS322 Database

	TW1		TW4		TW7	
Effective.			(TW2,TW3)		(TW5,TW6)	
Measure	R	I	R	I	R	I
Precision	.188	.181	.216	.212	.207	.216
	.125	.123	.136	.133	.137	.142
Q	19	22	14	15	16	17
	15	14	9	9	10	9

For both databases the Q values of reclustering (C^3M) and incremental clustering (C^2ICM) are comparable. For the TODS322 case, C^2ICM outperforms C^3M in terms of precision based on the matching function TW7. For the INSPEC database again the Q values for both algorithms are comparable. The precision observations for both algorithms are very close to each other. The maximum difference is observed for TW1: The CBR precision of C^2ICM is 3.5 percent lower than that of C^3M (with $d_s = 10$). On the other hand for TW2 and TW3 with $d_s = 10$, the precision of our incremental clustering is slightly better than that of reclustering. All the differences are less than ten percent; therefore, can be considered insignificant [2].

In [7] we showed that C^3M outperforms various clustering methods (i.e., single link, complete linkage, average link, Ward method), which are currently used in the IR literature, by 47.5 percent on the average. The results of this study indicate that the incremental clustering algorithm C^2ICM provides a retrieval quality comparable with that of reclustering using C^3M and therefore outperforms the other algorithms.

6. CONCLUSION

Clustering of very large document databases is a necessity to reduce the space/time complexity of information retrieval and to provide the capability of browsing documents which are similar to the relevant documents. The periodic updating of clusters is required due to the dynamic nature of document databases. An algorithm for incremental clustering, C^2ICM, has been introduced. Its complexity analysis, the cost comparison with reclustering, and an analytical analysis of the expected clustering behavior in dynamic IR environments are provided. To judge the effectiveness of the approach, experiments were designed to test the similarity between the clusters generated by C^2ICM and those clusters generated by reclustering the entire document database. In the experiments two different databases were used, one of which is a common database, INSPEC, containing 12684 documents.

Using various measures we showed that the clusters generated by C^2ICM held significant similarity with the ones generated by the reclustering using C^3M algorithm. It is also shown that this similarity is not by chance and the algorithm is cost effective with respect to reclustering. The experimental observations show that the algorithm can be used for large increments or for several steps. It is also demonstrated that the generated clustering structures are valid. For the validation process user judgements are used. It is shown that the average number of clusters accessed to retrieve all documents relevant to user queries is significantly smaller than that of random clustering. The experiments also showed that C^2ICM provides an effective retrieval environment. It is demonstrated that IR performance of the algorithm compares favorably with the performance of C^3M, which outperforms various clustering methods currently used in the IR literature. The results of this study indicate that the incremental clustering algorithm provides retrieval quality equal to that of reclustering.

This study shows that the C^2ICM algorithm is an efficient and effective cluster maintenance algorithm for dynamic document databases. The remaining points to be covered are the implementation and testing of an extension of the algorithm for instant clustering which would be appropriate for office automation environments. Another related problem is the efficient and effective combination of CBR and inverted index-based FS using various matching functions. Currently, we are working on these problems. Two other research problems that we will undertake in near future is the use of clustering with hypertext technology and to develop some performance measurement methods for hypertext-CBR as an IR techniques.

APPENDIX: An Incremental Clustering Example

Preliminary Concepts and Initial Clustering: Let us apply the preliminary concepts of the algorithm on an example D matrix. Consider the D matrix of Figure A1.

$$D = \begin{array}{cccccc|c} t_1 & t_2 & t_3 & t_4 & t_5 & t_6 & \\ 1 & 0 & 0 & 1 & 0 & 1 & d_1 \\ 1 & 1 & 1 & 1 & 0 & 0 & d_2 \\ 1 & 0 & 0 & 0 & 1 & 1 & d_3 \\ 0 & 0 & 0 & 0 & 1 & 1 & d_4 \\ 1 & 0 & 1 & 0 & 0 & 0 & d_5 \end{array}$$

Figure A1. The example D matrix.

For example, from the formula of c_{ij},

$$c_{ij} = \alpha_i \times \sum_{k=1}^{n} (d_{ik} \times \beta_k \times d_{jk}) \qquad \text{where } 1 \leq i, j \leq m$$

the probability of selecting and term of d_1 from d_3 is

$c_{13} = 1/3 \times (1 \times 1/4 \times 1 + 0 \times 1 \times 0 + 0 \times 1/2 \times 0 + 1 \times 1/2 \times 0 + 0 \times 1/2 \times 1 + 1 \times 1/3 \times 1)$

$= 1/3 \times (1/4 + 1/3) = 7/36 \approx 0.194$

Repeating the process for all other c_{ij} values the C matrix of Figure A.2 is obtained. Notice that the entire C matrix is given for the sake of illustration. However, the implementation of the algorithm and the CC-based concepts do not require complete construction of the C matrix.

$$C = \begin{bmatrix} 0.361 & 0.250 & 0.194 & 0.111 & 0.083 \\ 0.188 & 0.563 & 0.063 & 0.000 & 0.188 \\ 0.194 & 0.083 & 0.361 & 0.277 & 0.083 \\ 0.167 & 0.000 & 0.417 & 0.417 & 0.000 \\ 0.125 & 0.375 & 0.125 & 0.000 & 0.375 \end{bmatrix}$$

Figure A2. The C matrix corresponding to the D matrix of Figure A1.

Then $n_c = \delta_1 + \delta_2 + \delta_3 + \delta_4 + \delta_5 + \delta_6$ $(\delta_i = c_{ii})$, $n_c = 0.361 + 0.563 + 0.361 + 0.417 + 0.375 = 2.077 \approx 2$. Next find the seed powers, p_i $1 \leq i \leq 5$ (refer to Section 3 for the definition).

$p_1 = 0.361 \times (1 - 0.361) \times 3 = 0.692$

$p_2 = 0.563 \times (1 - 0.563) \times 4 = 0.984$

$p_3 = 0.361 \times (1 - 0.361) \times 3 = 0.692$

$p_4 = 0.417 \times (1 - 0.417) \times 2 = 0.486$

$p_5 = 0.375 \times (1 - 0.375) \times 2 = 0.469$

We select d_2 and d_1 as the cluster seeds, since they have the highest seed powers and are not identical. Only the seed powers have to be checked in this case since their difference is greater than the threshold (in the experiments the threshold used is 0.001). Notice that d_1 and d_3 have the same seed power. But they are not identical documents/seeds ($c_{11} = c_{33}$, $c_{13} = c_{31}$, but $c_{11} \neq c_{13}$, $c_{33} \neq c_{31}$, see the 5th property of the C matrix, Section 3); hence, both of them are eligible as a seed and the choice between them is arbitrary. Before continuing, we must state that, this arbitrary choice is possible for only the last seed; and therefore, it is unimportant for large document databases.

The IISD (Inverted term Index for Seed Documents) of the example D matrix is shown in Figure A.3. Notice that for a binary D matrix the weight information is redundant.

t_1 --> [<d_1, 1>, <d_2, 1>]

t_2 --> [<d_2, 1>]

t_3 --> [<d_2, 1>]

t_4 --> [<d_1, 1>, <d_2, 1>]

t_5 --> [nil]

t_6 --> [<d_1, 1>]

Figure A.3. The IISD for the example D matrix (the seeds are d_1 and d_2).

Now we have to determine which of the nonseed documents $\{d_3, d_4, d_5\}$ will be assigned to which of the seeds, $\{d_1, d_2\}$. So we need to calculate the c_{ij} values for i= 3, 4, 5; and j= 1, 2. To do this we only have to sum terms that are common to d_i and d_j. This is achieved using the data structure IISD. To cluster d_3, we first set $c_{31} = 0$ and $c_{32} = 0$, then consider each nonzero term of d_3 document vector to traverse the IISD. During the traversal of t_1 list c_{31} and c_{32} are incrementally updated as follows.

$c_{31} = c_{31} + \alpha_3 \times (d_{31} \times \beta_1 \times d_{11}) = 0 + 1/3 \times (1 \times 1/4 \times 1) = 1/12$

$c_{32} = c_{32} + \alpha_3 \times (d_{31} \times \beta_1 \times d_{21}) = 0 + 1/3 \times (1 \times 1/4 \times 1) = 1/12$

The term lists for t_2, t_3, and t_4 are bypassed since these terms do not appear in d_3. Similarly t_5 does not appear in the seed documents. The last term of d_3 is t_6 and its effect on c_{31} and c_{32} is computed as follows.

$c_{31} = c_{31} + \alpha_3 \times (d_{36} \times \beta_6 \times d_{16}) = 1/12 + 1/3 \times (1 \times 1/3 \times 1) = 1/12 + 1/9 = 0.194$
Document 2 does not contain t_6 and c_{32} remains the same and $c_{32} = 1/12 = 0.083$. Since $c_{13} > c_{32}$, the document d_3 will be clustered with d_1. Continuing in this manner we find that the two clusters are $C_1 = \{d_1, d_3, d_4\}$ and $C_2 = \{d_2, d_5\}$.

Incremental Maintenance: Now assume we want to delete documents d_2 and d_5. Notice this removes all the documents containing term t_3. Also, we want to add documents d_6, d_7, and d_8, as shown in Figure A4. Notice also that the new documents have introduced the new terms t_7 and t_8 to the database.

$$D = \begin{array}{c c} \begin{array}{ccccccc} t_1 & t_2 & t_4 & t_5 & t_6 & t_7 & t_8 \end{array} & \\ \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \end{bmatrix} & \begin{array}{c} d_1 \\ d_3 \\ d_4 \\ d_6 \\ d_7 \\ d_8 \end{array} \end{array}$$

Figure A4. The D matrix for the updated database.

Then Figure A5 is the resulting C matrix. In this matrix the document numbers are explicitly specified since there is no correspondence between row/column positions and document numbers.

$$C = \begin{array}{c c} \begin{array}{cccccc} d_1 & d_3 & d_4 & d_6 & d_7 & d_8 \end{array} & \\ \begin{bmatrix} 0.289 & 0.178 & 0.067 & 0.178 & 0.111 & 0.178 \\ 0.178 & 0.289 & 0.178 & 0.067 & 0.222 & 0.067 \\ 0.100 & 0.267 & 0.267 & 0.100 & 0.167 & 0.100 \\ 0.133 & 0.050 & 0.050 & 0.342 & 0.208 & 0.217 \\ 0.067 & 0.133 & 0.067 & 0.167 & 0.500 & 0.067 \\ 0.178 & 0.067 & 0.067 & 0.289 & 0.111 & 0.289 \end{bmatrix} & \begin{array}{c} d_1 \\ d_3 \\ d_4 \\ d_6 \\ d_7 \\ d_8 \end{array} \end{array}$$

Figure A5. The C matrix for the updated database.

Next we find the number of clusters for this database:

n_c = 0.298 + 0.289 + 0.267 + 0.342 + 0.500 + 0.289

= 1.994 $\approx$ 2.

Then the seed powers are (in decreasing order): p_7 = 1.250, p_6 = 0.915, p_3 = 0.666, p_8 = 0.616, p_1 = 0.563, p_4 = 0.469. Since n_c = 2 we examine the top two seed powers and discover that they are not equivalent, so we choose the documents d_7 and d_6 to be the seeds.

Continuing with the algorithm, we notice that since d_2 was a seed and was deleted, the cluster associated with d_2, C_2, from the previous step must be falsified; the documents in that cluster (excluding any deleted documents) must be reclustered. Also, since we see that d_1, which was a seed in the first step, is no longer a seed, we must falsify C_1 and recluster all nondeleted documents in C_1. As can be seen, all the remaining documents have to be clustered again. With larger databases the number of falsified clusters is much smaller in comparison to the number of nonfalsified clusters (see Figure 2 in Section 5.2).

Now we have to determine which of the nonseed documents {d_1, d_3, d_4, d_8} will be assigned to which of the seeds, {d_7, d_6}. So we need to calculate the c_{ij} values for i= 1, 3, 4, 8; and j= 7, 6. (For brevity the IISD is not shown.) For d_1, c_{16}= 0.178 and c_{17} = 0.111 since $c_{16} > c_{17}$, the document d_1 will be clustered with d_7. Continuing in this manner the resulting clusters are C_1 = {d_3, d_4, d_7} and C_2 = {d_1, d_6, d_8}.

REFERENCES

1. Anderberg, M. R. *Cluster Analysis for Applications.* Academic Press, New York, 1973.

2. Belkin, N. J., Croft, W. B. "Retrieval Techniques." In *Annual Review of Information Science and Technology, ARIST.* Vol. 22, M. E. Williams, Ed. Elsevier Science, Amsterdam, The Netherlands, 1987, 109-145.

3. Can, F., Ozkarahan, E. A. "A Dynamic Cluster Maintenance System for Information Retrieval." In *Proceedings of the 10th Annual International ACM-SIGIR Conference* (New Orleans, LA, June 1987). ACM, New York, 1987, 123-131.

4. Can, F., Ozkarahan, E. A. "Dynamic Cluster Maintenance." *Information Processing and Management.* 25, 3 (1989), 275-291.

5. Can, F. "Validation of Clustering Structures in Information Retrieval." In *Proceedings of the Canadian Conference on Electrical and Computer Engineering* (Montreal, Quebec, September 1989). EIC, Montreal, Quebec, 1989, 572-575.

6. Can, F., Drochak II, N. D. "Incremental Clustering for Dynamic Document Databases." In *Proceedings of the 1990 Symposium on Applied Computing* (Fayetteville, AR, April 1990). IEEE, Los Alamitos, CA, 1990, 61-67.

7. Can, F., Ozkarahan, E. A. "Concepts and Effectiveness of the Cover-Coefficient-Based Clustering Methodology for Text Databases." *ACM Transactions on Database Systems*. 15, 4 (December 1990), (to appear).

8. Crouch, D. B. "A File Organization and Maintenance Procedure for Dynamic Document Collections." *Information Processing and Management*. 11 (1975), 11-21.

9. Defays, D. "An Efficient Algorithm for Complete Link Method." *The Computer Journal*. 20 (1977), 364-366.

10. Dialog. *Dialog Database Catalog*. Dialog Information Services Inc.,1989.

11. El-Hamdouchi, A., Willett, P. "Comparison of Hierarchical Agglomerative Clustering Methods for Document Retrieval." *The Computer Journal*. 32, 3 (June 1989), 220-227.

12. Faloutsos, C. "Access Methods for Text." *ACM Computing Surveys*. 17, 1, (March 1985), 49-74.

13. Griffiths, A., Luckhurst, C., Willett, P. "Using Interdocument Similarity Information in Document Retrieval Systems." *Journal of the American Society for Information Science*. 37, 1 (1986), 3-11.

14. Hall, J. L. *Online Bibliographic Databases: A Directory and Sourcebook, 4th ed.* Aslib, Great Britain, 1986.

15. Heaps, H. S. *Information Retrieval Computational and Theoretical Aspects*. Academic Press, New York, 1978.

16. Jain A. K., Dubes, R. C. *Algorithms for Clustering Data*. Prentice Hall, Englewood Cliffs, NJ, 1988.

17. Maron, M. E. "Depth of Indexing." *Journal of the American Society for Information Science*. 30 (1979), 224-228.

18. Ozkarahan, E. A., Can, F. "An Automatic and Tunable Indexing System." In *Proceedings of the 9th Annual International ACM-SIGIR Conference* (Pisa, Italy, September 1986), ACM, New York, 234-243.

19. Salton, G. *Dynamic Information and Library Processing*. Prentice Hall, Englewood Cliffs NJ, 1975.

20. Salton, G., Wong, A. "Generation and Search of Clustered Files." *ACM Transactions on Database Systems*. 3, 4 (Dec. 1978), 321-346.

21. Salton, G., McGill, M. J. *Introduction to Modern Information Retrieval*. McGraw Hill, New York, 1983.

22. Salton, G., Buckley, C. "Term-Weighting Approaches in Automatic Text Retrieval." *Information Processing and Management*. 24, 5 (1988), 513-523.

23. Salton, G. *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer.* Addison Wesley, Reading, Massachusetts, 1989.

24. Van Rijsbergen, C. J. *Information Retrieval, 2nd ed.* Butterworths, London, 1979.

25. Voorhees, E. M. *The Effectiveness and Efficiency of Agglomerative Hierarchical Clustering in Document Retrieval.* PhD Dissertation. Dept. of Computer Science, Cornell University, Ithaca, NY, 1986.

26. Voorhees, E. M. "The Efficiency of Inverted Index and Cluster Searches." In *Proceedings of the 9th Annual International ACM-SIGIR Conference* (Pisa, Italy, September 1986), ACM, New York, 164-174.

27. Willett, P. "Recent Trends in Hierarchical Document Clustering: A Critical Review." *Information Processing and Management*. 24, 5 (1988), 577-597.

28. Yao, S. B. "Approximating Block Accesses in Database Organizations." *Communications of the ACM*. 20, 4 (April 1977), 260-261.

29. Yu, C. T., Chen, C. H. "Adaptive Document Clustering." In *Proceedings of the 8th Annual International ACM-SIGIR Conference* (Montreal, Quebec, June 1985). ACM, New York, 1985, 197-203.