Incremental Clustering for Dynamic Information Processing

FAZLI CAN
Miami University

Clustering of very large document databases is useful for both searching and browsing. The periodic updating of clusters is required due to the dynamic nature of databases. An algorithm for incremental clustering is introduced. The complexity and cost analysis of the algorithm together with an investigation of its expected behavior are presented. Through empirical testing it is shown that the algorithm achieves cost effectiveness and generates statistically valid clusters that are compatible with those of reclustering. The experimental evidence shows that the algorithm creates an effective and efficient retrieval environment.

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: Best-match cluster search, cluster validity, cover coefficient, dynamic information retrieval environment, information retrieval, information retrieval effectiveness, information retrieval efficiency

1. INTRODUCTION

This work differs from our previous research [8, 9] because it studies multiple maintenance steps with a 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 (Cover-Coefficient-based Incremental Clustering Methodology) suitable for very large dynamic document databases (for examples see Table I).

Section 2 reviews the clustering and cluster maintenance problem. In Section 3 the base concept of the incremental clustering algorithm, the cover coefficient (CC) concept, and its relation to clustering are described. Section 4 introduces the incremental clustering methodology, C^2ICM, its complexity and cost analysis, and an analytical analysis of its expected behavior in

This work was supported by a MUCFR grant. Author's address: Department of Systems Analysis, Miami University, Oxford, OH 45056. email: fc74sanf@miamiu.bitnet

Table I. File Size and Update Characteristics of Some Commercial Document Databases [11]

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.

dynamic document database environments. Section 5 covers our experimental design and evaluation. The experiment is based on the INSPEC database of 12,684 documents and 77 queries.

2. CLUSTERING AND CLUSTER MAINTENANCE

Three basic categories of clustering methodologies are *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 link" 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.

Iterative algorithms try to embellish a clustering structure according to an optimization function. The generation of the initial clusters can be done, for example, by using seed-oriented clustering [1].

The problem of cluster maintenance deals with the modification of clustering structures due to the addition of new documents or deletion of old documents (or both). A close examination of clustering algorithms reveals that most of them are unsuitable for cluster maintenance [26, p. 58]. In the literature there are very few maintenance algorithms. In general, these algorithms are developed for growing databases; most of them, however, can also be used for document deletion.

The *cluster-splitting* approach for maintenance 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 [24]. The major problems with this approach [21,

p. 494] are: (1) the generated clustering structure depends on the order of processing of the documents; and (2) it starts to degenerate with relatively small increases (such as 25% to 50%) in the size of the database.

Adaptive clustering is based on user queries. The method introduced in [31] 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 the documents relevant to it are shifted closer to each other. To ensure that all documents are not bunched up together, the centroid of these documents is moved away from the centroid of the entire database. This approach has some disadvantages: (1) the clusters 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) convergence can be slow. The applicability of such algorithms in dynamic environments 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 —and this may be very costly. The update cost of the single-link method is reasonable; its IR effectiveness is known to be unsatisfactory, however [12, 27, 29]. The time and space requirements of the group average and the complete link approaches are prohibitive, since they require the complete knowledge of similarities among old documents [27, pp. 29–30].

Another possibility for handling database growth is to use a cheap clustering algorithm of *mlogm* complexity to recluster the entire database after document additions and deletions. However, even though some algorithms have theoretical complexity of *mlogm*, the experimental evaluations have shown that, due to the large proportionality constant, the actual time taken for clustering is greater than that of an m^2 algorithm [26, p. 59]. These points imply that an efficient maintenance algorithm would be better than reclustering the whole database.

3. PRELIMINARY CONCEPTS

The base of C^2ICM, the CC (cover-coefficient) concept, provides a measure of similarities among documents (see also [10]). This 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 to generate a partitioned clustering structure. (To aid in reading, the definition of the notation introduced in Sections 3 and 4 is provided in Table II.)

The CC concept determines document relationships in a multidimensional term space by a two-stage probability experiment. It maps an $m \times n$ (document by term) D matrix [10] into an $m \times m$ C, *cover coefficient*, matrix. The individual entries of C, $c_{ij}(1 \leq i, j \leq m)$, indicate the probability of selecting any term of d_i from d_j and are defined as follows:

$$c_{ij} = \alpha_i x \sum_{k=1}^{n} (d_{ik} \times \beta_k \times d_{jk})$$

Table II. Symbol Notation Introduced in Sections 3, 4

Symbol	Definition / Meaning
c_{ij}	Extent to which d_i is covered by d_j (according to CC concept)
d_c	Average cluster size, $\max(1, m / n) \le d_c \le m$
D_r	Set of documents to be clustered during an increment
k	No. of incremental updates
m (D_m)	No of documents (Current set of documents)
m_1	Initial no. of documents before all increments
m' ($D_{m'}$)	No. of added documents in an increment (Set of such documents), $D_{m'} \subseteq D_r$
m'' ($D_{m''}$)	No. of deleted documents in an increment (Set of such documents)
n (n_k)	No. of terms (No. of terms after kth increment)
n_c ($n_{c,k}$)	No. of clusters (No. of clusters after kth increment), $1 \le n_c \le \min(m, n)$
p_i	Seed power of d_i
r	$\|D_r\| / m'$ ($r \ge 1$)
R	No. of reclustered documents in k increments
t	Total no. of nonzero entries in D
t_g	Average no. of documents per term, term generality (t / n)
t_{gs}	Average no. of seed documents per term
x_d	Average no. of distinct terms per document, depth of indexing (t / m)
α_i	Reciprocal of ith row sum of D
β_j	Reciprocal of jth column sum of D
δ_i	Decoupling coefficient of d_i ($\delta_i = c_{ii}$, $0 < \delta_i \le 1$)
ψ_i	Coupling coefficient of d_i ($\psi_i = 1 - \delta_i$, $0 \le \psi_i < 1$)
δ'_j, ψ'_j	Like δ_j and ψ_j, but for t_j

where α_i and β_k are the reciprocals of the ith row sum and kth column sum, respectively. To apply the above formula, each document must have at least one term and each term must appear at least in one document. By definition, the entries of the D matrix can be any nonnegative real number.

The two-stage probability experiment behind c_{ij} can be better explained by an intuitive analogy [17, p. 94]. Suppose we have many urns (d_i contains many terms), and also suppose that each urn contains balls of different colors (the terms of d_i may appear in many documents). Then what is the probability of selecting a ball of a particular color (i.e., d_j)? The experimental computation of this probability first requires the selection of an urn at random and second involves an attempt at drawing the ball of that particular color from the selected urn. In terms of the D matrix, in the first stage, from the terms (urns) of d_i we choose any term, t_k, at random; in c_{ij} this random selection is provided by the product $\alpha_i \times d_{ik}$. In the second stage, from the selected term (urn), t_k, we try to draw d_j (the ball of that particular color); in c_{ij} the product $\beta_k \times d_{jk}$ signifies the probability of selecting d_j. In this experiment, we have to consider the contribution of each urn (terms of d_i) to the selection probability of a ball of that particular color (d_j); that is why we have the summation for all terms in the definition of c_{ij}. Accordingly, in terms of D, this two-stage experiment reveals the probability of selecting any term of d_i from d_j. 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}.

Some characteristics of the C matrix follow [10, Table I]:

(1) $0 < c_{ij} \leq 1$, $0 \leq c_{ij} < 1$;
(2) $c_{ii} \geq c_{ij}$ and $\min(c_{ii}) = 1/m$ for a binary D matrix;
(3) $(c_{i1} + c_{i2} + \cdots + c_{im}) = 1$;
(4) $c_{ij} = 0 \leftrightarrow c_{ji} = 0$, $c_{ij} > 0 \leftrightarrow c_{ji} > 0$, and in general $c_{ij} \neq c_{ji}$;
(5) $c_{ii} = c_{jj} = c_{ij} = c_{ji} \leftrightarrow d_i$ and d_i are identical;
(6) d_i is distinct $\leftrightarrow c_{ii} = 1$.

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. 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 d_j covers d_i. It is also true that if two 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 other documents, we let $\delta_i = c_{ii}$ 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 $n \times n$ whose entries are defined as follows.

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

Using c'_{ij}, the same concepts of decoupling ($\delta'_j = c'_{jj}$), coupling ($\psi'_j = 1 - \delta'_j$), and number of clusters (see below) can be defined for the terms.

In a database with similar documents, the diagonal entries ($\delta_i s$) 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.

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

It is shown that for a given D matrix, the number of clusters indicated by documents (i.e., by $\delta_i s$) and terms (i.e., by $\delta'_j s$) is the same and thus $\max(n_c) = \min(m, n)$. Hence, the average document cluster size, d_c, is in the following range.

$$\max(1, m/n) \leq d_c \leq m.$$

We can see that n_c is a function of D, and is not an input parameter. This provides flexibility and adds robustness to the methodology. Notice that unique documents and documents with high decoupling coefficient values should have their own clusters. Accordingly, for the rest of the clustering process n_c should be decreased by the summation of the decoupling coefficients of such documents. In practical settings, however, it may be hard to find unique documents. In the experiments, we did not observe any unique documents, and in our study we assume that such documents will be put into a ragbag cluster.

In forming the initial clusters, we employ C^3M Cover-Coefficient-based Clustering Methodology [10]. Our incremental clustering algorithm, C^2ICM, is an extension of C^3M. Both algorithms produce partitions, and their clusters "grow" from seed documents.

In a seed-oriented approach, a cluster seed must be able to attract some nonseed documents around itself and at the same time it must be separated from other seeds as much as possible. To satisfy these constraints, we introduce the cluster *seed power* concept. The cluster seed power p_i of d_i is defined as follows:

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

corresponding to the binary and weighted version of the D matrix, respectively. Here, δ_i enables separation of clusters through document dissimilarities and ψ_i provides intracluster cohesion through document similarities. The third term, the summation, provides normalization. In a weighted D matrix, we need to pay attention to the weights of terms in a document. By the second equation, a term with a high weight in d_i, but with a skewed frequency (i.e., a very frequent or a very rare term), will not contribute significantly to the seed power of d_i. Thereafter the documents with the n_c highest seed powers are selected as the clusters seeds. In a 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 100% of the time. However, if the seed powers of two candidate seed documents d_i and d_j are identical, we compare c_{ii}, c_{jj}, c_{ij}, and c_{ji} values. The equality of these four entities indicates that these documents are identical (refer to the fifth property of the C matrix), and therefore, only one of them can be used as a cluster seed. (For each eliminated seed, we consider another document from the sorted list of documents.)

Toward the end of the seed-selection process, the number of documents that are eligible as cluster seeds can be more than the number of documents needed to complete the selection process. If this condition occurs in practice, it would be for a small fraction of n_c; therefore, the selection can be done arbitrarily (or to generate a unique clustering structure, we may choose the documents with smaller numbers). However, to provide better coverage of the nonseed documents, the eligible documents that use the maximum number of terms which are not used by the previously chosen cluster seeds can be selected.

After determining n_c seed documents, for all remaining $(m - n_c)$ nonseed documents, we determine the cluster seed that 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. If there are more than one such seed documents, then it is assigned to the cluster whose seed has the lowest document number. That is, each document can be a member of only one cluster.

The extent to which a nonseed document d_i is covered by a seed document d_j is designated by c_{ij}. The calculation of c_{ij} involves summing many zero terms (products), 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. This term list contains ⟨document seed number, term weight⟩ pairs for each seed in which that term appears. This new data structure is referred to as the Inverted term Index for Seed Documents (IISD). Using the 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 [10]. 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 [10, 28].

4. THE INCREMENTAL CLUSTERING ALGORITHM

In this section we first introduce the C^2ICM algorithm, then discuss its complexity and cost analysis. Also included is a study of the effects of the database dynamism on the generated clustering structures.

4.1 The C^2ICM Algorithm

Incremental clustering starts with m_1 documents. These documents are clustered using the C^3M algorithm. After this, clusters are updated due to newcomers (additions) and obsolete documents (deletions) using the C^2ICM algorithm. In the rest of the paper the symbols m' and m'', respectively, indicate the number of added and 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 C^2ICM is given in the following; the symbols "$\cup$" and "$-$" indicate the set operations union and difference, respectively.

C^2ICM

[*a*] *Compute n_c and 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 $m' \gg m''$.)*

[*b*] *Determine D_r, the set of documents to be clustered. Cluster these documents by 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 ragbag cluster.*

[*d*] *Apply the above steps for each 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); or (2) one or more of its nonseed documents becomes a seed after a database update.

Notice that an old document that belongs to a nonfalsified enduring cluster might more strongly belong to another one after new seeds are computed. The solution of this problem—that is, the assignment of such documents to the most appropriate seed—makes the clustering structure generated by C^2ICM the same as the one generated by C^3M. However, this impedes the efficiency of the maintenance process and, therefore, is avoided. This is to say, the C^2ICM algorithm is guaranteed to be order-independent for the members of D_r only. This 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 C^3M, the implementation of C^2ICM 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 23,061 entries with a storage overhead of just 5.6% of the D matrix. (The D matrix of the INSPEC database contains 412,255 nonzero entries, see Section 5.2.)

To cluster the initial set of m_1 documents (before incrementing the database), we use C^3M. Its computational requirement is as follows [10]:

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

where x_d and t_{gs}, respectively, indicate average number of distinct terms per document and average number of seeds per term. (Actually, t_{gs} accounts only for terms that appear in two or more documents, since only such terms can show up both in seeds and nonseeds.) 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 $m_1 \gg n_c$ the difference of $(m_1 - n_c)$ is taken as m_1.

The cost of generating the IISD is ignored, since it is very low: $(n_c \times x_d)$. (In fact, the previous approximation and this one neutralize each other.) 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 $3 \ll t_{gs} \ll t_g$, 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 hierarchical clustering algorithms currently used in the IR literature, whose complexity range is from $O(m^2)$ to $O(m^3)$ [27, 29].

To make the analysis of C^{2}ICM simpler, we ignore document deletions; however, this does not invalidate the analysis, since $m' \gg m''$ in a typical environment.

The computational cost of the first update is a good representative of any incremental step, and is given below:

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

where r is the ratio of documents to be clustered ($|D_r|$) to number of added documents in an increment (m'). In the experiments we observed that the maximum value of r is equal to 2 ($r \geq 1$). In the above expression, the first term determines the complexity, since it is larger than the other two terms. Accordingly, the complexity of one incremental clustering step is $O(x_d \times (m_1 + m'))$.

The complexity of the algorithm cannot be reduced by saving the previous decoupling coefficients of documents and terms and the seed power values. In fact, additional research has shown that the magnitude of recomputations to update these data structures is as large as the calculation of their values from scratch.

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, that is, if all seeds are falsified, C^{2}ICM 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 be unable to reflect the effect of the additions/deletions on the old database. We want some reclustering to happen during incremental handling, but not too much.)

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

$$[(m_1) + (m_1 + m') + \cdots + (m_1 + (k-1) \times m')]$$
$$= [k \times m_1 + (k \times (k-1)/2) \times m'].$$

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 $2 \times R \ll [2 \times k \times m_1 + k \times (k - 1) \times m']$.

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 [10]. 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 in terms of number of clusters and average cluster size.

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

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

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. We assume that the average depth of indexing, x_d, is the same throughout different stages of a database. This is an experimentally verified assumption, as will be shown in Section 5.2. Notice that individual documents may show considerable divergence from the average, and our assumption imposes nothing on the variance.

4.4.1 *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 $k-1$th and kth 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:

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

Accordingly, the estimated (total) number of clusters at 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 indexing vocabulary (V), n_{k-1}, and the current size of 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 [16, p. 206]. Therefore, the number of clusters grows indefinitely, but its rate of growth decreases as the database size increases.

In this environment, if many documents are added, though in general they use the existing vocabulary terms only, it would be desirable to increase n_c because very large clusters defeat the purpose of clustering. To solve this problem, we may apply C^3M within large clusters. Since the size of individual clusters would be much smaller than the size of the database, the cost incurred by this extra clustering would be reasonable.

4.4.2 *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 a 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:

$$d_c \begin{cases} > x_d & \text{if} \quad m > n \\ < x_d & \text{if} \quad m < n \end{cases}$$

5. THE EXPERIMENTS

Our experiments were designed to

(1) observe the effect of a dynamic environment of 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; and
(6) show the efficiency of CBR.

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

5.1 The Document Database

In this study we use the INSPEC database. INSPEC contains 12,684 documents covering topics in computer science and electrical engineering. The D matrix and the queries of the INSPEC database are in common with other studies [10, 12, 23, 27, 28]. Additional experimental results for a smaller database, TODS322, are available in [5, 7].

5.2 The Clustering Experiments

This section verifies analytical results (predictions about number of clusters and average cluster size and the cost analysis of the algorithm) with experimental evidence. For this purpose we created six experiment cases by

Table III. 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	4014	5748	7482	9216	10950	12684

Table IV Characteristics of the D Matrices

Matrix No	n	t	t_q	x_d
1	8009	128284	16.02	31 96
2	9637	183459	19 04	31.92
3	11074	242569	21 90	32.42
4	12452	301446	24 21	32 71
5	13537	356867	26.36	32 59
6	14573	412255	28 29	32.50

choosing different values for m_1, while keeping the same size of increments. No deletions were done. We used cases where $m_1 = 32\%$, then 45%, 59%, 73%, and 86% of the database. We added documents to the initial m_1 in steps of about 14% of the full size of the database (1734 documents). The actual numbers of documents are shown in Table III. For example, if we begin with matrix 1, this provides us with five increments (steps). The database grows from 4,014 documents with increments of size 1734, and matrix 6 contains all documents available in the collection.

Throughout the experiments, we used dynamic indexing, i.e., the documents of each increment are allowed to use new terms that have not been used by the old documents. That is, matrices 1 through 5 are subsets of the complete D matrix (matrix 6). The tests were done both on weighted and binary D matrices. The results of both cases are similar; the weighted version however, always gives a more robust clustering structure. In this paper we only report the results of the weighted cases, since a weighted D matrix is more general than a binary one.

The characteristics of the D matrices of the experiments are shown in Table IV. Table IV shows that for a given D matrix, x_d, the 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.

If we begin with matrix 1, we know that there are five incremental steps. The number of clusters for each step (k) is shown in Table V. (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 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 (given by $(m/n) \times x_d$) values are very close to the corresponding actual d_c values. These results demonstrate that we can estimate the

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

Step (k)	n_c	Est. n_c	d_c	Est. d_c
0	269	*	15	16
1	321	324	18	19
2	364	369	21	22
3	407	409	23	24
4	442	443	25	26
5	475	476	27	28

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

Table VI. Incremental Clustering Behavior of the Algorithm

	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

values of n_c and d_c by the main variables of the document database m and n. This information is valuable to predict the future requirements of an IRS in terms of secondary storage.

The incremental clustering behavior of the algorithm is given in Table VI. 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% increase in database size, $m'/m_1 = 1734/4014 = 0.43$) falsifies 22% (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 $m_1 = 4014$ (for these cases $m = 7482$). 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. (Notice that in the experiments, for a given m value, no matter how many incremental steps have been performed, we have the same D matrix as the input, and therefore the same set of documents is selected as the cluster seeds.) accordingly, they both falsify the same number of clusters, actually the same seeds but not necessarily the clusters containing exactly the same set of documents. This can be seen from the total number of falsified documents.

In the experiments, the case $m_1 = 4014$ provides the maximum number of incremental steps and reflects the behavior of the algorithm for other values of m_1; it therefore deserves more attention. With $m_1 = 4014$, we have five

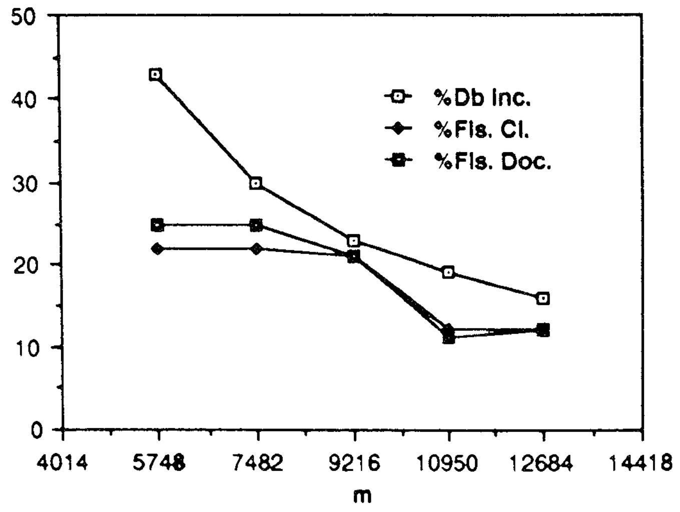

Fig. 1. The behavior of the algorithm ($m_1 = 4014$, $m' = 1734$, for five increments)

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 1 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 decreases and the cost effectiveness of the algorithm increases. This is an expected behavior, since smaller increments would be unlikely to significantly impact an already existing clustering structure of a relatively large database.

5.3 The Similarity Experiments

The purpose of the similarity experiments is to check how well the maintenance approach is generating clusters that are compatible with those of reclustering while attaining lower cost. For this purpose we measure the similarity between the clusters generated by incremental clustering (C^2ICM) and the clusters generated by reclustering (C^3M).

The measurement of similarity is done using the "corrected" Rand (CR) coefficient [18, p. 175]. The CR coefficient has a maximum value of one when the clustering structures (partitions) compared are identical, and a value of zero when they are generated by chance (i.e., CR is corrected for chance).

The CR values for matrix 1 to matrix 5 are 0.57, 0.58, 0.63, 0.70, and 0.79, respectively. (As expected, increase in matrix number or decrease in the number of incremental steps implies higher similarity.) These results show that the similarity between the clustering structures of C^2ICM and C^3M did not, in a statistical sense, happen by chance.

5.4 Validation of Clustering Structures

One must show that the clustering structure is a good representative of the intrinsic character of the data set being clustered. Such a structure is called valid. The two other cluster validity issues, 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 provided in [18] and [4] and [29], respectively.

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 that 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. 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 [10] of Yao's theorem [30] for estimating number of block accesses. After obtaining the n_t and n_{tr} values, 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 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 gives the n_t and n_{tr} values for all incremental clustering experiments. The characteristics of the queries are given in Section 5.5.3. To show that n_t is significantly less than n_{tr}, we must know the probability density function (baseline distribution) of n_{tr}. For this purpose, we generated 1000 random clustering structures for each clustering structure produced by C^2ICM and obtained the average number of target clusters for each random case, $n_t(r)$. For example, for matrix 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 less than the random case, since all of the $n_t(r)$ observations have a value greater than n_t. The same is observed in all of the incremental clustering experiments. This shows that the clusters are not an artifact of the C^2ICM algorithm, but are indeed 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 [10] 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 link, average link, and Ward method [12]. It is also shown that CBR

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

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

performance of C^3M is compatible with a very demanding (in terms of storage and CPU time) implementation of a complete link method [28]. 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 taken as 4014, and the clustering structure obtained after the fifth increment is used in the retrieval experiments (see Table III). Notice that this clustering structure is the outcome of the most stringent conditions of our experimental environment.

5.5.1 *Evaluation Approach.* In this study, for retrieval we use the *best-match CBR* policy. That is, in CBR, clusters are first ranked according to similarity of their centroids with the user query. Then, n_s clusters are selected and d_s documents of the selected clusters are chosen according to their similarity to the query. The selected clusters and documents must have nonzero similarity to the query. Typically, a user evaluates the first 10 to 20 documents returned by the system and after that is either satisfied or the query is reformulated. In this study, d_s values 10 and 20 are used.

The effectiveness measures used in this study are the average precision[1] for all queries and total number of queries with no relevant documents, Q.

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 has three components: the term frequency component (TFC), the collection frequency component (CFC), and the normalization component (NC) [23]. 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.

$$similarity\ (d, q) = \sum_{k=1}^{n} w_{dk} \times w_{qk}.$$

Salton and Buckley [23] obtained 287 unique combinations of document/query term-weight assignments. In the same study, six of these combinations are recommended due to their superior IR performance. In this study we used

[1] *Precision* is defined as the ratio of the number of retrieved relevant documents to the number of retrieved documents.

Table VIII. Characteristics of the Queries

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
77	15 82	33 03	2543	1940	577

these six matching functions in addition to the cosine matching function. Each of these matching functions enables us to test the performance of C^2ICM under a different condition.

For brevity, 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 [10], they are again referred to as TW1 through TW7; in [23] 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 of INSPEC are presented in Table VIII. The query terms are weighted; the weight information is used whenever needed.

The cluster representatives (centroids) are constructed using the terms with the highest total number of occurrences within the documents of a cluster. The maximum centroid length (i.e., the number of different terms in a centroid), which is a parameter, is set to 250. This value is just 1.72% of the total number of distinct terms used for the description of the database. In our previous research we showed that, after some point, the increase in centroid length does not increase the effectiveness of IR, and the above centroid length is observed as suitable for INSPEC [10]. Similar results are also obtained for hierarchical clustering of various document databases [28].

The characteristics of the centroids produced for C^3M and C^2ICM are very similar to each other, as shown in Table IX. In this table, x_c indicates the average number of distinct terms per "centroid"; $\%n$ indicates the percentage of terms that appear in at least one centroid; t_{gc} indicates the average number of centroids per term for the terms that appear in at least one centroid; and $\%D$ indicates the total size of the centroid vectors as a percentage of t in the corresponding D matrix.

The storage cost of the centroids is less than one third of that of the D matrix (refer to column $\%D$). This 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 and 52, respectively, for C^3M and C^2ICM. However, as stated earlier, we know that longer centroids do not result in superior retrieval effectiveness. With centroid length 250, the percentage of cluster terms used in their centroids, $100 \times (x_c/$average number of distinct terms per "cluster"$) = \%x_c$, is 51 for both algorithms.

Table IX. Characteristics of the Centroids (the second row is taken from [10])

Algorithm	x_c	%n	t_{gc}	%D
C^2ICM	227 73	58	12.74	26
C^3M	237 09	60	12.96	27

Table X. Effectiveness of the Reclustering (R) and Incremental Clustering (I), for $n_s = 50$ (figures for R are taken from [10])

Effective Measure	TW1		TW2		TW3		TW4		TW5		TW6		TW7	
	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

5.5.4 *Retrieval Effectiveness.* In this section we compare the CBR effectiveness of C^2ICM and C^3M using the matching functions defined in Section 5.5.2 and show that the former is as effective as the latter, which is known to be very effective for CBR.

For the implementation of the best-match CBR, we first have to decide about the number of clusters to be selected, n_s. In the best-match CBR process, the retrieval effectiveness increases up to a certain n_s. After this "saturation" point, it either remains the same or increases very slowly [10, Figure 6]. In our experiments the saturation point for INSPEC is observed for $n_s = 50$, i.e., when 11% of the clusters are selected ($n_c = 475$). This percentage is typical for best-match CBR [21, p. 376]. The results of the IR experiments for $n_s = 50$ are shown in Table X. The first and second rows of each effectiveness measure (precision, Q) are for d_s values of 10 and 20, respectively.

Table X shows that Q values of reclustering (R) and incremental clustering (I) are comparable. Similarly, 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% 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 incremental clustering is slightly better than that of reclustering. All the differences are less than 10%, and can therefore be considered insignificant [2].

The results of this section indicate that the incremental clustering algorithm C^2ICM provides a retrieval effectiveness quality comparable with that of reclustering using C^3M; thus it is superior to the algorithms that are outperformed by C^3M.

Given that the foundation of the clustering methodology is CC (with the seeds that it produces), these seeds are also used as cluster centroids. This approach represents a cluster with a typical member, and is similar to the method defined in [25]. As expected, the use of a typical member (in this case the seed) of a cluster as its representative is less effective (about 18% less in terms of precision). However, one direct advantage of this approach is that CBR is as efficient as FS, based on inverted index searches—moreover, in a dynamic environment, it completely eliminates the overhead of centroid maintenance.

5.6 Efficiency Considerations

In this discussion only the more crucial efficiency component, time, is considered. The storage efficiency of the retrieval environment is analyzed in [6].

As stated before, retrieval effectiveness reaches its saturation when $n_s = 50$. In other words, for INSPEC, 11% ($n_c = 475$) of clusters is selected, and about the same percentage of the documents is considered for retrieval. As we have stated previously, this percentage is typical for best-match CBR. However, it might be considerably greater than that of the case of hierarchical searches. The exact number of documents matched during hierarchical CBR is unavailable [12, 27, 28], but as shown in the following, the method used in this study is much more efficient than the top-down search [27, 28] that we used in our effectiveness comparison. We are unable to compare our results with that of the bottom-up search strategies reported in [12], since no efficiency figure is available for them.

Hierarchical searches usually involve the matching of query vectors with many centroid and document vectors that have little in common with the queries [22]. For example, the top-down search of the INSPEC database is performed on the complete link tree structure containing 11,878 clusters in 16 levels [27, p. 60]. The best-match CBR can be implemented very efficiently, independent of both the number of clusters to be selected (n_s) and the number of documents to be retrieved (d_s), as shown below.

In the conventional implementation of best-match CBR (for brevity we use CBR to indicate best-match CBR), query vectors are first matched with the inverted indexes of the centroids (IC) and then with the document vectors (DV) of the members of the selected clusters. This implementation of CBR is referred to as ICDV because of its search components. In ICDV, the query vectors are only matched with the centroids that have common terms with the query vectors, but the same is not true for the DV component. Notice that the generation of separate inverted indexes for the members of each cluster is impractical. However, we can still introduce a new implementation strategy and exploit the efficiency of the inverted index search (IIS) of the complete database in CBR. In this new approach, which is referred to as ICIIS because of its components, the clusters are again selected using IC; the documents of the selected clusters are then ranked using the results of the IIS performed on the *complete database*! By definition, the efficiency of the ICIIS approach is independent of n_s and d_s. The ICIIS strategy contains some unproductive computation in its IIS component, since only a subset of the database is the member of the selected clusters. This drawback is compromised by the shortness of the query vectors: in IR it is known that query vectors are usually short [10, 14, 23].

Table XI. Average Query Processing Time Using Different Retrieval Techniques

Retrieval Technique	Average Query Processing Time (s)
IIS (FS)	1.53
ICIIS (best-match CBR)	2 41
ICDV (best-match CBR), $n_s = 50$	5 31
Top-down Search on Complete Link Hiearc. [28]	13 07

The efficiency of the search techniques IIS, ICIIS, and ICDV is measured in terms of average query processing time using the storage and retrieval environment model defined in [27, 28]. In our CBR environment, this model can easily accommodate one million documents [3, 6]. The average query processing times of IIS, ICIIS, ICDV, and the top-down search (taken from [28]) on the INSPEC database are provided in Table XI. In this table the ICDV result is given in terms of the matching function TW2. The efficiency of IIS and ICIIS is independent of the matching function. In [6] it is shown that the effect of matching function on the efficiency of ICDV is negligible (less than 5%), thus the ICDV result of Table XI is typical. The matching function of the top-down search is similar to TW6 [10]. In all of the experiments, including the top-down search, maximum centroid length is 250. The experiments show that after a threshold the efficiency of ICIIS is unaffected by the centroid length [6]. This is because the terms brought by a longer centroid are not necessarily the query terms that have not been selected by using a shorter centroid length or shorter centroids may contain all of the query terms. The same (i.e., independence of efficiency from the centroid length) is almost true for ICDV [6], but is false for the top-down search [28].

The results of this section show that C^2ICM creates an efficient IR environment using ICIIS. Table XI shows that IIS is the most efficient method and requires 1.53s. The average query processing times using ICIIS and ICDV are, respectively, 2.41s and 5.31s. The same average for the top-down search is 13.07s. In other words, the top-down search requires 442% more processing time with respect to the ICIIS search performed on the clustering structure generated by C^2ICM. As an aside, the average query processing times for ICIIS and ICDV on the clustering structure generated by C^3M are, respectively, 2.41s and 4.82s. Notice that for C^2ICM and C^3M, ICIIS values are practically the same (due to rounding, the values are exactly the same) and ICDV values are very close to each other. This stems from the similarity of the clustering structures generated by the algorithms.

6. CONCLUSION

Clustering is useful for both searching and browsing of very large document databases. The periodic updating of clusters is required due to the dynamic nature of document databases. In this study 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. The experimental observations show that the algorithm creates an effective and efficient retrieval environment, and it is cost effective with respect to reclustering and can be used for many increments of various sizes.

ACKNOWLEDGMENTS

The author would like to acknowledge the constructive criticism and helpful suggestions made by the three anonymous referees and R. B. Allen. Thanks also to N. D. Drochak II, to D. J. Anderson for programming some parts of the experimental environment, to E. A. Ozkarahan for his comments on an earlier draft of this paper, and to J. M. Patton for helpful discussion.

REFERENCES

1. ANDERBERG, M. R. *Clustering Analysis for Applications*. Academic Press, New York, 1973.
2. BELKIN, N. J., AND CROFT, W. B. Retrieval techniques. *Ann. Rev. Inf. Scι. Technol. ARIST. 22*, (1987), 109–145.
3. BUCKLEY, C., AND LEWITT, A. F. Optimization of inverted vector searches. In *Proceedings of the 8th Annual International ACM-SIGIR Conference* (Montreal, Que., June 1985). ACM, New York, 1985, 97–110.
4. CAN, F. Validation of clustering structures in information retrieval. In *Proceedings of the Canadian Conference on Electrical and Computer Engineering* (Montreal, Que., Sept., 1989). EIC, Montreal, Que., 1989, 572–575.
5. CAN, F. Experiments of incremental clustering. Working Paper 91-002, Dept. of Systems Analysis, Miami Univ., Oxford, Oh., Aug. 1991.
6. CAN, F. On the efficiency of best-match cluster searches. (Submitted for publication.)
7. CAN, F., AND DROCHAK II, N. D. Incremental clustering for dynamic document databases. In *Proceedιngs of the 1990 Symposium on Applied Computing* (Fayetteville, Ark., April 1990). IEEE, Los Alamitos, Calif., 1990, 61–67.
8. CAN, F., AND OZKARAHAN, E.A. A dynamic cluster maintenance system for information retrieval. In *Proceedings of the 10th Annual Internatιonal ACM-SIGIR Conference* (New Orleans, La., June 1987). ACM, New York, 1987, 123–131.
9. CAN, F., AND OZKARAHAN, E. A. Dynamic cluster maintenance. *Inf. Process. Manage. 25*, 3 (1989), 275–291.
10. CAN, F., AND OZKARAHAN, E. A. Concepts and effectiveness of the cover-coefficient-based clustering methodology for text databases. *ACM Trans. Database Syst. 15*, 4 (Dec. 1990), 483–517.
11. DIALOG. *Dialog Database Catalog*. Dialog Information Services Inc., 1989.
12. EL-HAMDOUCHI, A., AND WILLETT, P. Comparison of hierarchical agglomerative clustering methods for document retrieval. *Computer J. 32*, 3 (June 1989), 220–227.
13. FALOUTSOS, C. Access methods for text. *ACM Comput. Surv. 17*, 1 (Mar. 1985), 49–74.
14. GRIFFITHS, A., LUCKHURST, C., AND WILLETT, P. Using interdocument similarity information in document retrieval systems. *J. Am. Soc. Inf. Sci. 37*, 1 (1986), 3–11.
15. HALL, J. L. *Online Bιbliographic Databases: A Directory and Sourcebook, 4th ed.* Aslib, Great Britain, 1986.
16. HEAPS, H. S. *Informatιon Retrieval Computational and Theoretical Aspects*. Academic Press, New York, 1978.
17. HODGES, J. L., AND LEHMANN, E. L. *Basic Concepts of Probabιlιty and Statιstics*. Holden-Day, San Fransisco, Calif., 1964.
18. JAIN, A. K., AND DUBES, R. C. *Algorithms for Clustering Data*. Prentice–Hall, Englewood Cliffs, N.J., 1988.
19. JARDINE, N., AND VAN RIJSBERGEN, C. J. The use of hierarchical clustering in information retrieval. *Inf. Storage Retrιeval 7* (1971), 217–240.
20. NIELSEN, J. *Hypertext and Hypermedia*. Academic Press, San Diego, Calif., 1990.

21. SALTON, G. *Dynamic Information and Library Processing*. Prentice-Hall, Englewood Cliffs, N.J , 1975.
22. SALTON, G. *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*. Addison Wesley, Reading, Mass., 1989
23. SALTON, G., AND BUCKLEY, C. Term-weighting approaches in automatic text retrieval *Inf. Process. Manage. 24*, 5 (1988), 513-523
24. SALTON, G , AND WONG, A. Generation and search of clustered files. *ACM Trans Database Syst. 3*, 4 (Dec. 1978), 321-346.
25. VAN RIJSBERGEN, C. J. The best-match problem in document retrieval. *Commun. ACM 17*, 11 (Nov. 1974), 648-649.
26. VAN RIJSBERGEN, C. J. *Information Retrieval, 2nd ed.* Butterworths. London, 1979.
27. VOORHEES, E. M. The effectiveness and efficiency of agglomerative hierarchical clustering in document retrieval. Ph.D. dissertation., Dept of Computer Science, Cornell Univ., Ithaca, N Y., 1986.
28. VOORHEES, E. M. The efficiency of inverted index and cluster searches. In *Proceedings of the 9th Annual International ACM-SIGIR Conference* (Pisa, Sept. 1986), ACM, New York, 164-174
29. WILLETT, P. Recent trends in hierarchical document clustering: A critical review. *Inf. Process. Manage. 24*, 5 (1988), 577-597.
30. YAO, S. B. Approximating block accesses in database organizations. *Commun. ACM 20*, 4 (April 1977), 260-261.
31 YU, C. T , AND CHEN, C. H Adaptive document clustering. In *Proceedings of the 8th Annual International ACM-SIGIR Conference* (Montreal, Que., June 1985), ACM, New York, 1985, 197-203.

Received December 1990; revised August 1992; accepted November 1992