Efficiency and Effectiveness of Query Processing in Cluster-Based Retrieval

FAZLI CAN[1]
İSMAİL SENGÖR ALTINGÖVDE
ENGİN DEMİR
(canf, ismaila, endemir)@cs.bilkent.edu.tr
Computer Engineering Department, Bilkent University
Bilkent, Ankara, 06533, Turkey; August 9, 2002

Abstract
Our research shows that for large databases, without considerable additional storage overhead, cluster-based retrieval (CBR) can compete with the time efficiency and effectiveness of the inverted index-based full search (FS). The proposed CBR method employs a storage structure that blends the cluster membership information into the inverted file posting lists. This approach significantly reduces the cost of similarity calculations for document ranking during query processing and improves efficiency. For example, in terms of in-memory computations, our new approach can reduce query processing time to 39% of FS. The experiments confirm that the approach is scalable and system performance improves with increasing database size. In the experiments, we use the Cover Coefficient-based Clustering Methodology (C^3M), and the Financial Times database of TREC-4 containing 210,158 documents of size 564 MB defined by 229,748 terms with total of 29,545,234 inverted index elements. This study provides CBR efficiency and effectiveness experiments using the largest corpus in an environment that employs no user interaction or user behavior assumption for clustering.

Keywords: Clustering, cluster-based retrieval, information retrieval, performance, query processing.

1. Introduction

The well-known *clustering hypothesis* states that "closely associated documents tend to be relevant to the same request." It is this hypothesis that motivates clustering of documents in a database [van Rijsbergen, 1979]. In the IR research, clustering has been originally introduced with the expectation of increasing the efficiency and effectiveness of the retrieval process [Jardine, van Rijsbergen, 1971; Salton, 1975].

In best-match cluster-based retrieval (CBR), it is assumed that there is a flat (one-level) clustering structure. In this environment, the queries are first compared with the clusters, or more accurately with the cluster representatives called centroids. Detailed query by document comparison is performed only within the selected clusters. In hierarchical (multi-level) clustering structures, it is possible to implement top-down or bottom-up cluster-based search strategies [Salton, McGill, 1983; Salton 1989; van Rijsbergen, 1979].

[1] Corresponding author, on sabbatical leave from Computer Science and Systems Analysis Department, Miami University, Oxford, OH 45056, USA. Present mail information, e-mail: canf@cs.bilkent.edu.tr, voice: +90 (312) 290-2613, fax: +90 (312) 266-4047. Permanent mail information, e-mail: canf@muohio.edu, voice: +1 (513) 529-5950, fax: +1 (513) 529-1524.

As with any such algorithm, the efficiency of CBR is important. In addition to being efficient, CBR should be effective in the sense of meeting user needs. Users may employ the clustering structure in exploring the document space to locate items close to some known documents. This is called *browsing*. Documents within a cluster can also be stored in close proximity to each other within a disk medium to minimize I/O delays [Salton, McGill, 1983, pp. 222-227]. However, it is shown that the operational effectiveness of many clustering algorithms is low [Shaw et al., 1997]. It is also observed that the efficiency of CBR is less than the efficiency of full search, FS, i.e., inverted index search of all documents [Can, 1994; Salton, 1989; Voorhees, 1986a, 1986b].

In this paper we study the efficiency and effectiveness of query processing in large clustered document collections using the best-match strategy. In the experiments we use the Cover Coefficient-based Clustering Methodology, C^3M, which we have introduced in our previous work. It is known that C^3M has superior performance with respect to other algorithms of the literature [Can, Ozkarahan, 1990] and can be used in dynamic environments in an incremental manner for cluster maintenance [Can, 1993]. The contributions of this study are the following.

- It shows that for large databases it is possible to perform CBR with an efficiency and effectiveness level, which is comparable with that of FS without considerable additional storage overhead. The CBR method proposed in this paper employs a simple and novel storage structure that blends the cluster membership information with the inverted file posting lists. As will be shown this approach significantly reduces the cost of similarity calculations during query processing. Our new storage structure is generic and clearly applicable with other clustering algorithms.
- It provides CBR experiments using the largest corpus in an environment with no user interaction or user behavior assumption. In the experiments we use the Financial Times database of the TREC-4 containing *210,158* documents of text size *564 MB* defined by *229,748* terms with total of *29,545,234* inverted index elements. For example, the studies reported in [Hearst, Pedersen, 1996; Silverstein, Pedersen, 1997] provide experiments with larger corpora; however, in their evaluations they assume that the user picks the optimal cluster or try to generate refined clusters with user interaction from an existing global clustering structure. In our work, all decisions are made automatically using similarity measures based on a global clustering structure with no user interaction.

There are various optimization techniques used for inverted index searches [Buckley, Lewit, 1985; Persin, 1994; Brown 1995; Moffat, Zobel, 1996]. They aim to use only the most informative parts of inverted list and try to increase efficiency of query processing without

deteriorating retrieval effectiveness. Such techniques can be employed during query processing to further improve the query optimization provided by CBR; however, search optimization (or pruning) with techniques other than clustering is beyond the scope of this study.

The rest of the paper is organized as follows. Section 2 explains our clustering algorithm C^3M and the file structures that can be used for the implementation of CBR including our new approach. Section 3 describes the experimental environment in terms of document database and queries. Section 4 covers the experimental results and their discussion. Section 5 reviews the previous and related work. The conclusions and pointers for future research are given in Section 6. A list of frequently used symbols and acronyms is provided in Table I.

Table I. Expanded form and meaning of frequently used acronyms and symbols

Acronym	Expanded Form
C^3M	Cover Coefficient-based Clustering Methodology
CBR	Cluster Based Retrieval
CVDV*	Centroid Vector Document Vector
CVIIS*	Centroid Vector Inv. Index Search
ICDV*	Inverted Centroid Document Vector
ICIIS*	Inverted Centroid Inverted Index Search
ICsIIS*	Inverted Centroid skip Inv. Index Search
FS	Full Search
FT	Financial Times database of TREC-4
FTs FTm	FT small and FT medium size versions

Symbol	Meaning
d_c	Average no. of documents per cluster
d_s	No. of documents selected during information retrieval
m	No. of documents
n	No. of terms
n_c	No. of clusters
n_s	No. of selected best matching clusters
n_t	Average no. of target clusters per query
n_{tr}	Average no. of target clusters per query in random clustering
t_g	Term generality (avg. no. of documents per term)
x_d	Depth of indexing (avg. no. of terms per document)

* CBR implementation method.

2. Clustering Algorithm and File Structures for CBR Implementation

In this section we briefly explain our clustering algorithm, the file structures that can be used for CBR, and our new CBR file structure that blends the clustering information with the traditional inverted file.

2.1 Clustering Algorithm: C^3M

As we indicated earlier in the experiments we use the C^3M algorithm, which is known to have good information retrieval performance. The C^3M algorithm assumes that the operational environment is based on the vector space model. Using this model, a document collection can be abstracted by a document, D, matrix of size m by n whose individual entries, d_{ij} ($1 \leq i \leq m$, $1 \leq j \leq n$), indicate the number of occurrences of term j (t_j) in document i (d_i).

Determining the number of clusters in a collection is a difficult problem [Jain, Dubes, 1988]. In other clustering algorithms, if it is required, the number of clusters, n_C, is usually a user specified parameter; in C^{3}M it is determined by using the cover-coefficient (CC) concept [Can, Ozkarahan, 1990; Yu, Meng, 1998]. In C^{3}M some of the documents are selected as cluster seeds and non-seed documents are assigned to one of the clusters initiated by the seed documents. According to CC, for an m by n document matrix the value range of n_C and the average cluster size (d_C) are as follows.

$$1 \leq n_C \leq \min(m, n); \ \max(1, m/n) \leq d_C \leq m$$

In C^{3}M, the document matrix D is mapped into an m by m cover-coefficient (C) matrix using a double-stage probability experiment. This asymmetric C matrix shows the relationships among the documents of a database. Note, however, that the implementation of C^{3}M does not require the complete C matrix. The diagonal entries of C are used to find the number of clusters, n_C, and the selection of cluster seeds. During the construction of clusters, the relationships between a non-seed document (d_i) and a seed document (d_j) is determined by calculating the c_{ij} entry of C, where c_{ij} indicates the extent with which d_i is covered by d_j. Therefore, the whole clustering process implies the calculation of (m + (m - n_C) x n_C) entries of the total m^2 entries of C. This is a small fraction of m^2, since $n_C << m$ (for some database examples please refer to Table II in Section 3.1). A thorough discussion and complexity analysis of C^{3}M are available in [Can, Ozkarahan, 1990].

The CC concept reveals the relationships between indexing and clustering [Can, Ozkarahan, 1990]. The CC-based indexing-clustering relationships are formulated as follows.

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

In these formulas, the meanings of the variables not used in the text so far are as follows.

d_C: m/n_C, average number of documents per cluster,

t: total number of non-zero entries in D matrix,

t_g: t/n, average number of different documents a term appears (term generality),

x_d: t/m, average number of distinct terms per document (depth of indexing).

These relationships can be used to predict the clustering structure of the algorithm.

It is shown that the algorithm can be used in a dynamic environment in an incremental fashion and such an approach saves clustering time and generates a clustering structure comparable to that of cluster regeneration by C^{3}M [Can, 1993; Can et al., 1995].

C^{3}M is a non-overlapping (partitioning) type clustering algorithm. In this paper, we introduce a new version of C^{3}M that creates overlapping clusters to test the effects of overlapping on efficiency and effectiveness of CBR. In the overlapping version a document

can be assigned to more than one cluster. For this purpose we slightly modified C^3M: Let c_{ij} be the CC value used to cluster d_i, i.e., let us assume that d_i joins to the cluster initiated by d_j since d_j provides the highest coverage of d_i among all cluster seeds. In the overlapping version we define a tolerance thres*h*old h ($1 > h > 0$) and assign d_i also to the cluster initiated by seed document d_k if $c_{ik} \geq h \times c_{ij}$. Furthermore, non-seed documents can be assigned to a preset maximum number of clusters at the most. We intuitively set the tolerance threshold to 0.9 and the maximum to five clusters.

2.2 File Structures for the Implementation of CBR

2.2.1 Previous File Structures for the Implementation of CBR

The (best-match) CBR search strategy has two components (a) selection of n_s number of best matching clusters using centroids, (b) selection of d_s number of best matching documents of the selected best matching clusters. For item (a) we have two file structure possibilities: CV (centroid vectors), and IC (inverted index of centroids). For item (b) we again have two possibilities: DV (document vectors), and IIS (inverted index of *all* documents). One remaining possibility for (b), a separate inverted index for the members of each cluster, is ignored due to its excessive cost in terms of disk accesses (for a query with k number of terms it would involve k disk accesses for each selected cluster). Hence, possible combinations of (a) and (b) define the following CBR implementation policies: CVDV, ICDV, CVIIS, ICIIS.

CVDV	Use **C**entroid **V**ectors for cluster selection and **D**ocument **V**ectors for document selection.
ICDV	Use **I**nverted index of **C**entroids for cluster selection and **D**ocument **V**ectors for document selection.
CVIIS	Use **C**entroid **V**ectors for cluster selection and **I**nverted **I**ndex **S**earch for document selection.
ICIIS	Use **I**nverted index of **C**entroids for cluster selection and **I**nverted **I**ndex **S**earch for document selection.

Figure 1. Summary of possible file structure strategies for CBR implementation (adapted from [Can, 1994]).

As summarized in Figure 1, CVDV means that for cluster match use centroid vectors as they are and for document selection from the selected clusters use the document vectors of the member documents. In ICIIS the documents of the best-matching clusters are selected using the results of FS, which is implemented by IIS. Notice that ICIIS is somewhat counter intuitive to the concept of CBR, since CBR considers only a subset of the database for retrieval purposes, but the IIS component of ICIIS will be performed on the complete database. However, ICIIS still has the potential of being efficient, since query vectors may contain a limited number of terms.

In [Can, 1994] the efficiency of these methods are measured in terms of CPU time, disk accesses, and storage requirements in a simulated environment defined in [Voorhees, 1986b]. The implementations from best to worst efficiency performance are ordered in the following way: ICIIS, ICDV, CVIIS, CVDV. It is observed that the ICIIS strategy is significantly better than the others. It is also shown that ICIIS is significantly better (*5.42* times faster) than a hierarchical cluster search technique, which is based on a complete link hierarchy [Can, 1994]. However, this earlier study has further revealed that ICIIS is inferior to FS (1.5 times slower) in terms of efficiency. In this study our aim is to introduce a CBR implementation strategy that would outperform ICIIS and achieve comparable efficiency and effectiveness with FS, and measure its performance in a large document collection.

2.2.2 The new CBR Implementation Using Skips

If we could generate a separate inverted index for the members of individual clusters, this would provide the most efficient computational environment for CBR. However, for a query with k terms if we select n_s best clusters this file structure implies (k x n_s) number of disk accesses, which is large since n_s would be large. To keep both the number of computations and number of disk accesses at its minimum, we have introduced a new CBR implementation structure that we call ICsIIS (IC skip IIS). In this structure IC has its usual structure; however, the IIS component stores not only the traditional posting list information but also the cluster membership information. In this organization posting list information associated with the members of a cluster are stored next to each other, and this is followed by those of the next cluster's. At the same time we keep a pointer from the beginning of one cluster sub-posting list to the next one. During query processing we use these pointers to skip the clusters, which are not selected as a best matching cluster. Another skips idea for efficient decompression of inverted indexes can be seen in [Moffat, Zobel, 1996].

An example file structure is provided in Figure 2 for a D matrix, which is clustered using C^3M. In this figure each posting list header contains the associated term, the number of posting list elements associated with that term, and the posting list pointer (disk address). The posting list elements are of two types, "cluster number – position of the next cluster," and "document number – term frequency" for the preceding cluster.

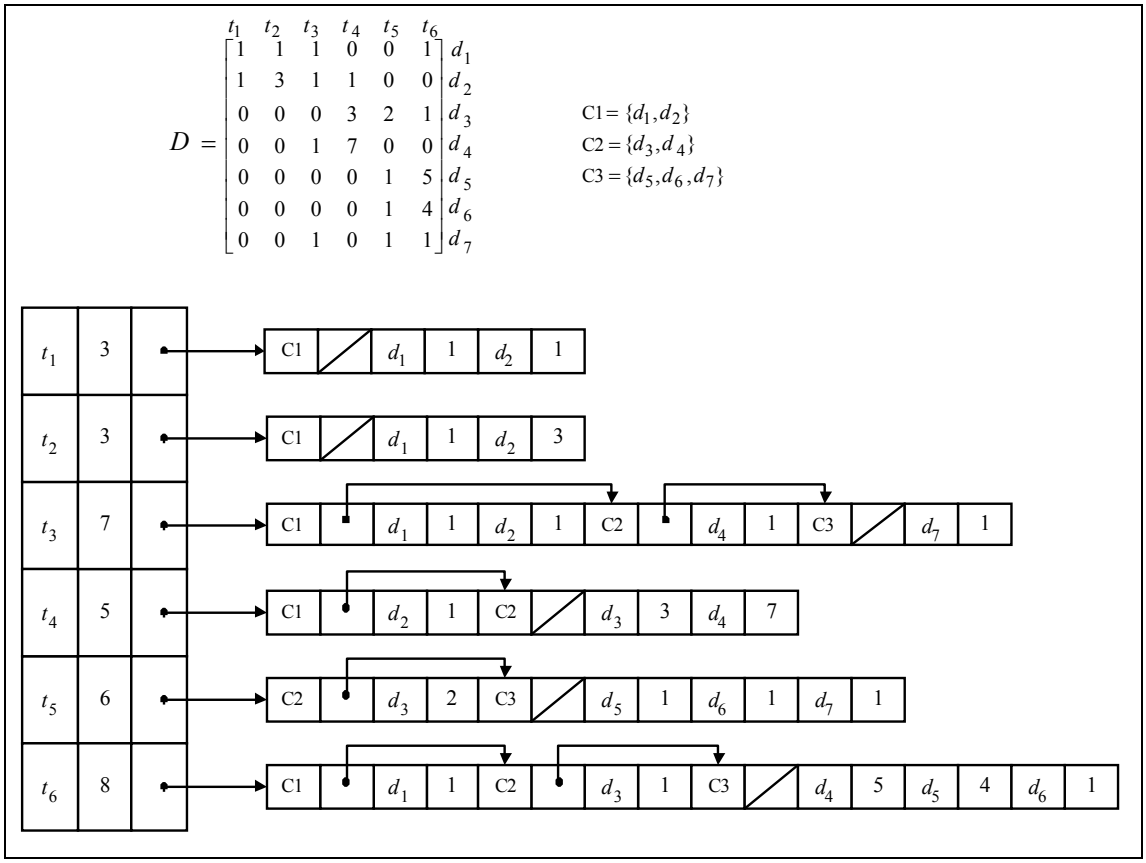


Figure 2. Example inverted file structure with skips.

Our skip structure is simple yet novel. In the previous CBR research a similar approach has not been used. For example, Salton and McGill's classical textbook [Salton, McGill, 1983, pp. 223-224] defines three cluster search strategies. Two of them are related to hierarchical cluster search and their concern is the storage organization of the cluster centroids. In the third CBR strategy, documents (not their inverted lists) are stored in cluster order, that is, one access to the "document file" retrieves a cluster of related documents. Our skip idea provides a completely new way of implementing CBR by clustering the individual posting lists elements. This is certainly different than accessing the "documents" in cluster order.

Salton wrote [Salton, 1989, p. 344]:

> *"In general, the efficiencies of inverted-file search techniques are difficult to match with any other file-search system because the only documents directly handled in the inverted-list approach are those included in certain inverted lists that are known in advance to have at least one term in common with the queries. In a clustered organization, on the other hand, many cluster centroids, and ultimately many documents, must be compared with query formulations that may have little in common with the queries."*

The CBR using the skip-based inverted index search technique overcomes the problem stated by Salton, i.e., it prevents matching many unnecessary documents with the queries. For example, in the clustering environment of Figure 2, if we assume that the user query contains the terms $\{t_3, t_5\}$ and the best matching clusters for this query are {C1, C3}, using the ICsIIS approach during query processing after selecting the best matching clusters we only consider the posting lists associated with t_3 and t_5. While processing the posting list of t_3 we skip the portion corresponding to C2 (since it is not a best matching cluster). Similarly, while processing the posting list of t_5, we again skip the unnecessary C2 portion of the posting list and only consider the part corresponding to C3. In other words, by using the skip approach we only handle the documents that we really need to match with the query.

In the implementation of the skip idea another alternative is to store the cluster number and skip information at the start of the posting lists. Here we adopt the former approach illustrated in Figure 2. Practically these two alternatives have no major difference in terms of posting list I/O time, since in almost all cases query term posting lists are read in their entirety because a term usually appears in enough number of different clusters that would require inputting its whole posting list. In query processing a significant portion of the time cost comes from similarity calculations for ranking, and skipping information helps us in considerably decreasing the cost of these calculations.

3. Experimental Environment

3.1 Document Database

In the experiments, Financial Times documents (FT database) of TREC-4 collection are used. The document database includes *210,158* newspaper articles published between 1991 and 1994. During the indexing stage, we eliminated English stop-words and numbers, and indexed the remaining words, and no stemming is used. The resulting lexicon contains *229,748* terms. The D matrix contains *29,545,234* non-zero elements. The average number of distinct terms per document, or depth of indexing x_d, is *140.6*, and the longest and shortest documents contain *3220* and *4* distinct terms, respectively. On the average each term appears in *128.6* different documents. This is the average number of distinct documents per term or term generality, t_g.

For easy reference statistical characteristics of the FT (TREC-4) collection are provided in Table II along with some other databases to give some sense of sizes of the important variables in traditional (INSPEC, NPL), and OPAC (BLISS, MARIAN) [Can et al., 1995; Kocberber et al., 1999] collections. In this table the number of clusters, n_c, is obtained by using C^3M. The numbers show that databases, more specifically their vector spaces, show various degrees of sparsity as indicated by the number of clusters. For example, FT (TREC-

4) is quite cohesive and the number of clusters is not that high. On the other hand, OPAC (library), BLISS-1 and MARIAN, vector spaces are sparse and contain relatively large number of clusters, since they cover documents in many different subject areas. The content cohesiveness of a database may be uniformly distributed and clusters may contain approximately the same number of documents or it can be skewed and it may contain a few number of large clusters containing relatively high number of related documents. We will revisit this issue later in Section 4.1 from our database's point of view.

Table II. Characteristics of the FT (TREC-4) and some Other Databases

Database	m No. of Documents	n No. of Terms	x_d Avg. No. of Distinct Terms/Doc.	n_c No. of Clusters	d_c Avg. No. of Docs./Clust.
BLISS-1*	152,850	166,216	25.7	6,468	25
MARIAN	42,815	59,536	11.2	5,218	8
INSPEC	12,684	14,573	32.5	475	27
NPL	11,429	7,491	20.0	359	32
FT(TREC-4)	210,158	229,748	140.6	1,640	128

* Approximate n_c value is calculated using the cover-coefficient-based formula: $n_c = n/x_d$.

3.2 Queries and Query Matching

We used the TREC-7 query topics corresponding to the TREC-4 collection (queries 351-400) along with their relevance judgments; on the average there are 38.1 relevant documents per query. In the experiments we used four different types of query sets first two of which are created from the TREC queries.

1. *Qshort* (short queries) created from the title field of the TREC queries, i.e., these are title-only queries.
2. *Qmedium* (medium length queries) created from the title and description fields (combined) of the TREC queries.
3. *Qlong,* created from the top retrieved document of each *Qmedium* query. We assume that, the relevance judgments of the original query also apply to them.
4. *Qgiant* created by combining a number of random documents from the original data set, and is used for the purpose of evaluating efficiency in its theoretical limits. For this single query we do not measure effectiveness since we have no relevance information for it.

Table III provides query sets summary information.

There are several query matching functions that depend on the term weighting used for document and query terms [Salton, Buckley, 1988]. In this study, the document term weights are assigned using the *term frequency* x *inverse document frequency* (IDF) formulation. While computing the weight of term t_j in document d_i, term frequency is computed as the number of occurrences of t_j in d_i, and IDF is *ln(number of all documents/number of*

documents containing t_j)+1. Once the term weights are obtained, document vector is normalized using cosine normalization [Salton, Buckley, 1988].

Table III. Query sets summary information
(last three columns indicate no. of terms)

Query Set	Source	Average	Min	Max
Qshort	TREC Query Titles	2.38	1	3
Qmedium	TREC Query Titles and Descriptions	8.16	2	19
Qlong	Top Relevant Document	190.04	13	612
Qgiant	Random Documents	2175.00	2175	2175

The term weights for query terms are calculated in a similar fashion to document term weights. In this case, for computing term frequency component, we use augmented normalized frequency formula defined as *(0.5 + 0.5* x *tf / max-tf)*. Here *max-tf* denotes the maximum number of times any term appears in the query vector. IDF component is obtained in exactly the same manner with the document terms. No normalization is done for query terms since it does not affect document ranking.

After obtaining weighted document (d) and query (q) vectors in an n dimensional vector space the query-document matching is performed using the following formula.

$$similarity(q,d) = \sum_{j=1}^{n} w_{qj} w_{dj}$$

The members of the best matching clusters (note that in CBR a subset of the entire collection is under consideration) are ranked according to their similarity to the query, and for the top 10 (20, 100) documents the effectiveness measures precision and recall are calculated. *Precision* is defined as the ratio of retrieved relevant documents to the number of retrieved documents, and *recall* is defined as the ratio of retrieved relevant documents to total number relevant documents in the collection.

4. Experimental Results

In this section, we present various experiments to compare the efficiency and effectiveness of three retrieval strategies: Full Search (FS), cluster based retrieval combined with a full inverted index (ICISS), and cluster based retrieval incorporating the skipping concept (IC*s*IIS). As stated before, it has been shown that ICISS is more efficient than some other CBR techniques in terms of paging and CPU time, but inferior to FS [Can, 1994]. In the following set of experiments, we first investigate the validity of C^3M clustering for the FT database, and then examine the effectiveness and efficiency of the three retrieval strategies (namely, FS, ICIIS, and ICsIIS) as we vary several environment parameters. Actually, ICIIS

and ICsIIS are the same in terms of their effectiveness since they are two different implementations of the same CBR operation; therefore, for these two we can only compare their efficiency. We also study the scalability of our results. In the rest of the paper, we use CBR interchangeably with ICIIS and ICsIIS when it is appropriate.

The experiments are performed on dual processor Pentium III 866 PC with 1 GB main memory and 20 GB SCSI HDD. The operating system installed on this PC is Windows NT 4.0™. The source code for the prototype implementation is available at http://www.cs.bilkent.edu.tr/~ismaila/ircode.htm.

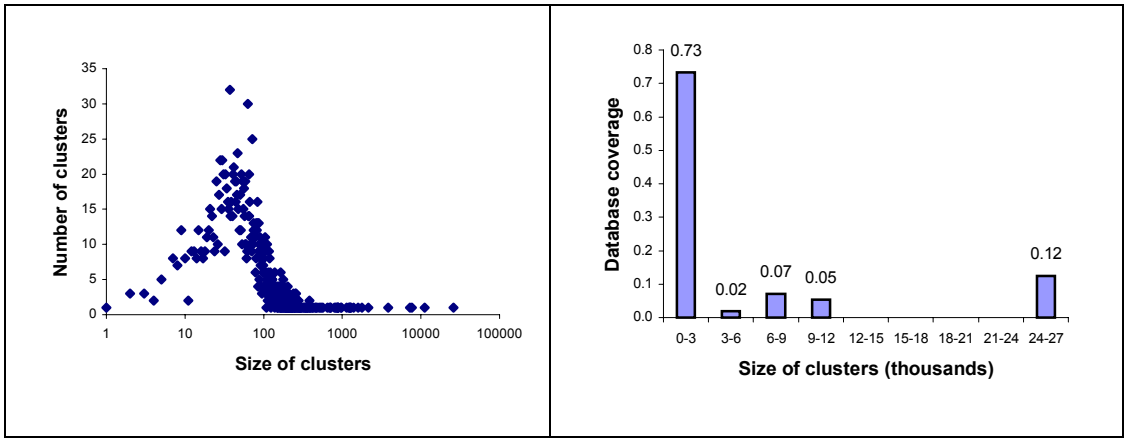
4.1 Clustering Structure: Generation, Characteristics and Validation

4.1.1 Cluster generation and characteristics of the generated clustering structure

Our experiments yield *1640* clusters (in both non-overlapping and overlapping cases) for the FT (TREC-4) collection. In the non-overlapping case the average cluster size is *128* (vs. *176* in overlapping), and the average number of distinct terms in a cluster is *4700* (vs. *5560*). Note that, in the overlapping case the total number of documents in the clusters is *288,685* (*vs. 210,158*), which means *33%* document duplication.

The generated clustering structure of the non-overlapping case follows the indexing-clustering relationships implied by the CC concept. For example, the indexing-clustering relationships $n_c = (m \times n)/t = m/t_g = n/x_d$, and $d_c = t_g$ are all observed in the experiments (for easy reference the values of these variables are repeated here, $m = 210{,}158$, $n = 229{,}748$, $t = 29{,}545{,}234$, $x_d = 140.6$, $t_g = 128.6$ and the values obtained for n_c and d_c after clustering are *1640* and *128*). For example, n_c was implied as *1634* by the relationships, which shows only a *%0.4* percent deviation from the real value obtained by actual clustering. Similarly, the d_c (*128*) value is almost identical with t_g. As shown in our related previous work [Can, 1990, 1993; Can et al., 1994] for a given D matrix the clustering structure to be generated by C^3M is predictable from the indexing characteristics of a database.

The size distribution of the clusters for the non-overlapping case is presented in Figure 3. In Figure 3.a the x-axis (in logarithmic scale) shows the cluster size in terms of documents and y-axis shows the number of clusters for the corresponding size. The figure shows that cluster sizes show variety, there are a few large clusters (largest one containing *26,076* documents) and some small clusters, and there are many clusters close to the average cluster size. Figure 3.b shows that majority of the documents (about *73%* of them) are stored in clusters with a size *1* to *3,000*. Please note that for only 10% of the queries top ten results include documents from the largest cluster, which means that our results are not significantly biased by the existence of a large cluster.



a. Cluster distributions in terms of no. of clusters per cluster size (logarithmic scale).

b. Ratio of total no. of documents observed in various cluster size windows.

Figure 3. Cluster size distribution information.

4.1.2 Validation of the generated clustering structure

Before using a clustering structure for IR we must show that it is significantly *different* from, or better than, random clustering in terms of reflecting the intrinsic nature of the data. Such a clustering structure is called valid. Two other cluster validity issues, clustering tendency and validity of individual clusters, are beyond the scope of this study [Jain, Dubes, 1988].

Our cluster validation approach is based on the users' judgment on the relevance of documents to queries and follows the methodology defined in [Can, Ozkarahan, 1990]. Given a query, a cluster is said to be a *target cluster* if it contains at least one relevant document to the query. Let n_t denote the *average number of target clusters for a set of queries*. Next, let's preserve the clustering structure and distribute all documents randomly to these clusters. The average number of target clusters for this case is shown by n_{tr} and its value can be calculated without creating random clusters by the modified form [Can, Ozkarahan, 1990] of Yao's formula [Yao, 1977]; however, for the validity decision we need the distribution of the 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 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}.

According to our validity criterion, we must know the probability density function of n_{tr}. For this purpose, we perform a Monte Carlo experiment and randomly distribute the documents to the cluster structure for *1000* times and for each experiment compute the average number of target clusters. The minimum, maximum, and average n_{tr} values are observed as *27.78*, *29.02* and *28.41* (see Figure 4 for the probability density function of the n_{tr} values). Then, we compute the n_t value, and it is *20.1*. Clearly, n_t is significantly different

than the random distributions n_{tr}, since it is less than all of the observed random n_{tr} values. These observations show that the clustering structure used in the retrieval experiments is not an artifact of the C^{3}M algorithm, on the contrary, significantly better than random and valid.

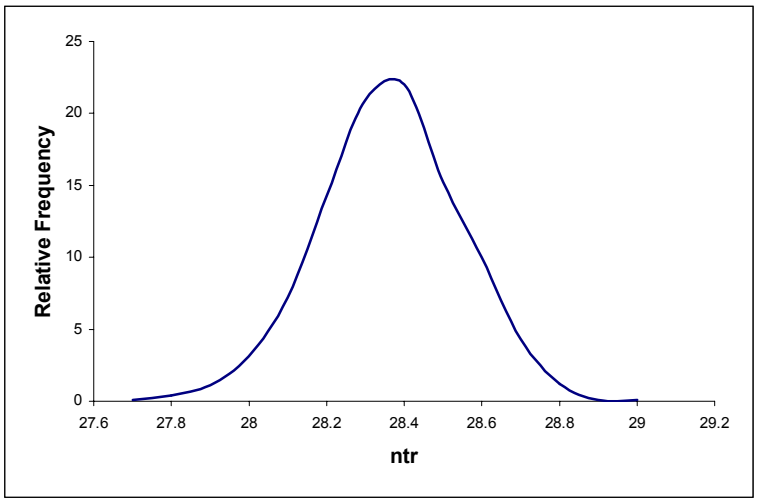


Figure 4. Histogram of n_{tr} values for the FT (TREC-4) database (n_t= 20.1).

4.2 Determining Number of Best Matching Clusters for CBR

The experiments show that selecting more clusters increases effectiveness since as we increase n_S (i.e., the number of selected clusters) more relevant documents would be covered [Salton, 1975, p. 376]. In our previous research, it is observed that effectiveness increases up to a certain n_S value, after this (saturation) point, the retrieval effectiveness remains the same or improves very slowly [Can & Ozkarahan, 1990, Figure 6]. For the INSPEC database, this saturation point is observed when n_S is about 10% of the clusters and during the related experiments about the same percentage of the documents is considered for retrieval. This percentage is typical for (best-match) CBR [Salton, 1975, p. 376].

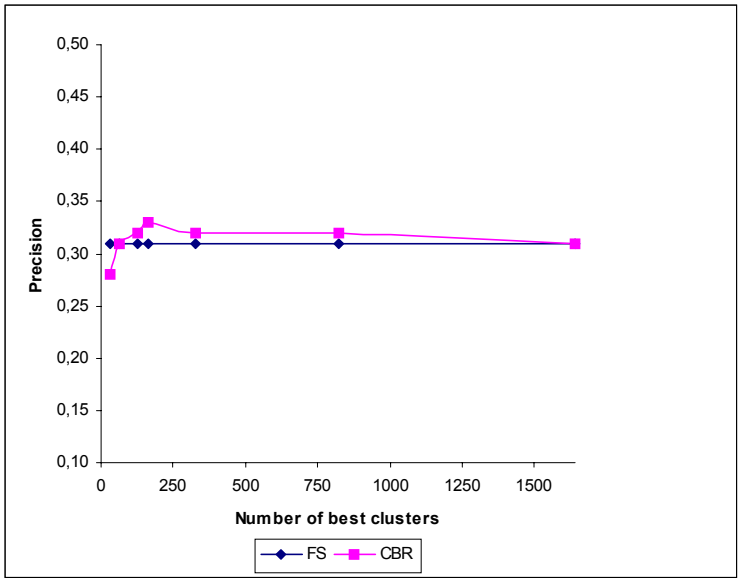


Figure 5. For d_s = 10 and query set *Qmedium*, mean average precision versus n_s.

In our experiments, for a range of n_s values, we retrieved top 10 documents for the query set *Qmedium* and measured the effectiveness in terms of mean average precision (i.e., average of the precision values observed when a relevant document is retrieved) [Baeza-Yates, Ribeiro-Neto, p. 80]. The results depicted in Figure 5 also confirm the above observation regarding INSPEC, where the effectiveness increases up to *164* clusters (10% of the cluster number n_c) and then no major change occurs. Therefore, we use 10% of n_c (n_s= 164 clusters) as the number of clusters to be used in the retrieval experiments.

No. (%) of Selected Clusters	Avg. No. (%) of Selected Documents
32 (1.95)	3 857.08 (1.84)
64 (3.90)	8 608.14 (4.10)
96 (5.85)	12 041.18 (5.73)
128 (7.81)	15 701.94 (7.47)
164 (10.0)	19 107.12 (9.09)
820 (50.0)	102 016.14 (48.54)
1 640 (100.00)	210 158.00 (100.00)

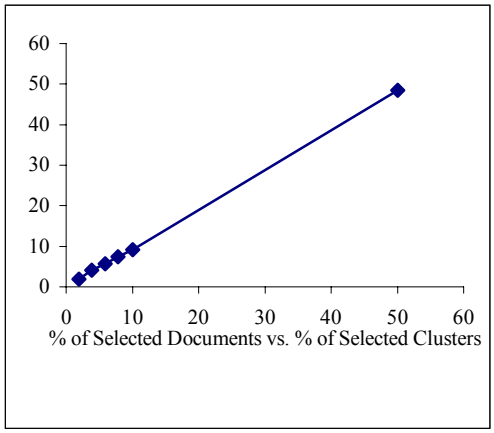


Figure 6. Relationship between number of selected clusters and number of documents in the selected clusters.

In Figure 6, we report the total number of documents in the clusters for each value of n_s. Both figures show that, for example, if we select the first best matching 164 clusters (10% of the existing clusters) we need to match 9.09% of the documents with the queries, since this much documents exists in the selected clusters (the numbers are averages for all queries). The observations show that there is a linear relationship between the percentage of clusters selected and the percentage of the database covered by them.

Determining the centroid terms is also an issue, since they may influence the effectiveness and efficiency of CBR. In this paper, the most frequent terms in clusters are chosen as centroid terms. The weight of a centroid term t_j is computed by *term frequency* x *IDF* formula, where term frequency is set to *1* and *IDF* is *ln (number of centroids / number of centroids including the term j) + 1*. In Sections 4.3 and 4.4, we use the ad-hoc centroid length value of 250 terms for both overlapping and non-overlapping cases. In Section 4.5, we further investigate the impact of various centroid length and term weighting strategies on the efficiency and effectiveness of query processing.

4.3 Effectiveness Experiments

To evaluate the effectiveness of three IR strategies, we retrieved the top 10, 20, and 100 documents for each of the query sets –namely, *Qshort*, *Qmedium* and *Qlong*–. The

experiments are conducted over both overlapping and non-overlapping clustering structures. The effectiveness results are presented by using both a TREC-like interpolated 11-point precision-recall graph [Baeza-Yates, Ribeiro-Neto, pp. 76-77] and a single mean average precision value (defined in the previos section) for each of the experiments. For the sake of saving space, we provide only top 10 effectiveness results for the experiments of the non-overlapping and overlapping clustering. For top 20 and 100 documents we have similar results.

Table IV. Mean average precision values for retrieval strategies (n_s = 164, d_s =10)

Query Set	FS	CBR (non-overlap.)	CBR (overlap.)
Qshort	**0.307**	0.296	0.268
Qmedium	0.314	0.326	**0.348**
Qlong	**0.383**	0.354	0.350

Table IV provides the mean average precision values for the retrieval strategies. For short queries, FS gives the best performance and it is followed by non-overlapping cases. In the case of medium size queries, CBR outperforms FS. For long queries, the reverse is true. For a more detailed comparison consider Figure 7. They illustrate that the effectiveness of FS and CBR are quite close to each other for different sets of queries with varying lengths. The effectiveness achieved over the overlapping cluster structure can be comparable or sometimes better than non-overlapping CBR and FS. For instance, Table IV shows that for *Qmedium*, non-overlapping CBR is better than FS, and overlapping CBR is even better than the non-overlapping case.

Table V. Effectiveness comparison of FS and CBR (ICIIS and ICsIIS), for non-overlapping clusters

Query Set	CBR = FS	CBR > FS	CBR < FS
Qshort	76%	6%	18%
Qmedium	70%	10%	20%
Qlong	88%	4%	8%

In Table V, for the same query sets and top 10 documents, we provide the effectiveness comparisons of individual queries during FS and CBR in non-overlapping case. For instance, CBR achieves better than FS in 6% of the *Qshort* queries. These results further indicate that there is no single best approach for IR, and either one of CBR or FS can perform better for different queries. Note that, our CBR approaches that blend inverted indexes with cluster based retrieval lead to new opportunities for combining the best results of both strategies, in a way that has not been done before. For example, during query processing we can handle query terms as in FS or CBR like a mixture depending on the query term properties.

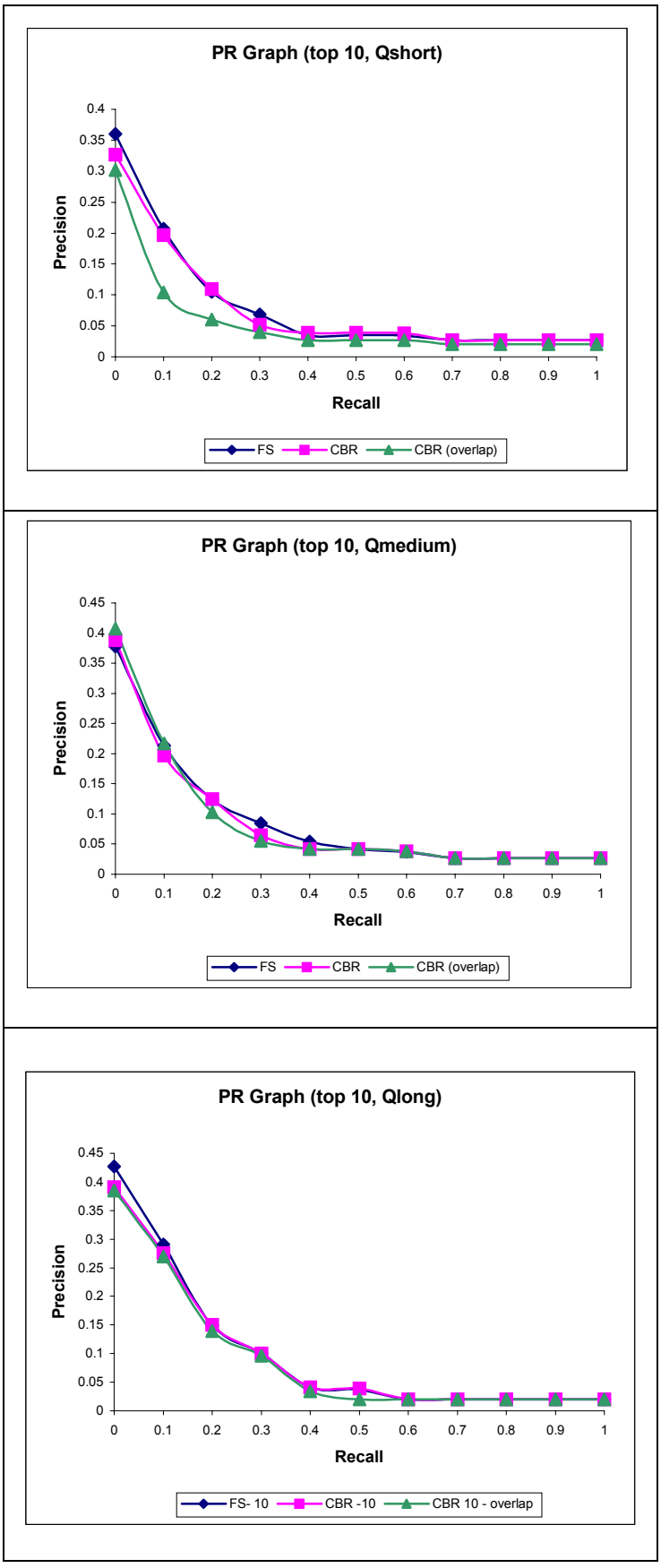


Figure 7. Interpolated PR graph for all query sets using top 10 (d_s= 10) documents.

4.4 Efficiency Experiments

4.4.1 Results in terms of processing requirements

We measure the efficiency of each retrieval strategy for top 10, 20 and 100 documents for all query sets. We evaluate the efficiency by using two different measures: (i) number of processed (accessed) posting list elements, and (ii) actual query processing time. In the following we provide the results for only non-overlapping case. For the overlapping case although the processing requirements are higher (due to the longer posting lists), the relative efficiency performance of compared algorithms does not exhibit a significant difference.

Table VI. Average number of document posting list elements processed by each retrieval strategy for each query set and percentage savings provided by ICsIIS

Query Set	FS and ICIIS	ICsIIS	% ICsISS savings wrt FS and ICIIS
Qshort	9,791	4,238	57
Qmedium	49,415	16,342	67
Qlong	1,813,734	784,005	57
Qgiant	12,398,355	6,637,800	47

In Table VI, we present the average number of document posting list elements processed for each query set while ranking documents using the query matching function. The posting lists brought to memory for ICsISS are longer than those for FS and ICIIS, as the skipping inverted index elements of ICsIIS include cluster information. On the other hand, in all cases, during the similarity calculations the ICsIIS strategy visits much less posting list elements than IIS and ICISS, since most of the posting list elements of ICsIIS are skipped due to our (skipping) storage structure. The last column shows the percentage savings provided by ICsIIS with respect to FS and ICIIS in posting list processing (the entries of this column are obtained from those of the second and third columns). The savings in terms of realistic query cases (*Qshort* to *Qlong*) savings range between 57% and 67%.

Table VII. Average in-memory processing time (sec.) per query for each retrieval strategy and relative performance of ICsIIS with respect to FS

Query Set	FS	ICISS	ICsIIS	ICsIIS/FS
Qshort	0.051	0.052	**0.038**	**0.75**
Qmedium	0.141	0.143	**0.055**	**0.39**
Qlong	1.090	1.107	**0.442**	**0.41**
Qgiant	4.319	4.385	**2.418**	**0.56**

The average in-memory processing time per query is reported in Table VII. The results reveal that the savings indicated in Table VI are proportionally reflected to the actual execution times. In all cases, ICsIIS performs faster than its competitors as the candidate result set to be considered is significantly reduced by the skipping technique proposed in this paper. Also note that, in our database the average posting list length (or term generality t_g) is

short, 129 elements. Our heuristics save more time for longer posting lists; therefore, we anticipate that efficiency results would be even better in databases with longer lists. This also explains why the savings for *Qshort* is relatively less than it may be expected.

Please note that, our skipping optimization is in-memory, whereas both ICIIS and ICsIIS have an extra cost of disk access for inverted centroid index entries. So, from a theoretical point of view CBR approaches discussed here suffer from this extra I/O cost. However, in practice, we observed that the extra I/O operations associated with accessing inverted centroid index entries are mostly compensated with today's file caching capabilities. In particular, the size of the inverted centroid index is only 1.5% of IIS (see Table IX of Section 4.4.2) and it can be effectively buffered or even totally stored in main memory. For instance, the average overall query processing times (in-memory computations + I/O overhead) for *Qmedium* is measured as 0.265, 0.301 and 0.240 seconds per query for FS, ICIIS and ICsIIS, respectively. In this case, all index structures are kept in the disk medium and extra I/O cost is reduced by OS buffering mechanism. Thus, we claim that ICsIIS is a worthwhile retrieval strategy also in terms of efficiency considerations.

4.4.2 Results in terms of storage requirements

As it is mentioned before, the ICsIIS strategy proposed in this paper incorporates cluster membership information into the inverted index posting lists. In Table VIII, we present statistics about the inverted index files stored on the disk for FS and the non-overlapping and overlapping clustering cases. It may be seen that the storage requirement for cluster-skipping inverted index is modestly higher than the ordinary inverted index file. The index creation time is 182 minutes for all structures (i.e., IIS, skip IIS non-overlapping, and skip IIS overlapping – the effect of skips on indexing time is negligible–). (Centroid generation time for both non-overlapping and overlapping structures is about 20 minutes.)

Table VIII. Storage requirements (size in MB) and posting list (PL) information for inverted index files

Inverted Index File	Size	Avg. Posting List Length (Docs/Term)	Max No. of Docs./PL	Min No. of Docs./PL
IIS	338	129	93,693	1
Skip ISS (non-overlapping)	426 (26% > FS)	162	95,329	2
Skip IIS (overlapping)	560 (65% > FS)	215	132,328	2

The storage requirements of FS (using IIS) is simply equal to "the total number of elements in the posting lists" (t) times "the size of an elements in the posting list." A posting list element consists of a 4 byte (integer) document number and 8 byte (double) term weight. There are 29,545,234 inverted index elements where each costs 12 bytes, leading to a total of 338MB.

In ICsIIS, posting lists include extra elements consisting of cluster number and the skip pointer. These additional elements also take 12 bytes to conform to the ordinary posting list elements (including document number and term weight). The average number of terms per cluster (*avg. terms/cluster*) is equal to 4700 and 5560, respectively, for the non-overlapping and overlapping clustering cases. This means that in the non-overlapping case the cluster number and the skip pointer address is included in 4700 different posting lists. This makes an additional cost of 88MB (4700 x 1640 (total number of clusters) x 12 bytes), i.e., total of 426 MB.

In the ICsIIS overlapping case the extra cost with respect to FS increases, *avg. terms/cluster* is equal to 5560; therefore, the cost due skip information is 104MB (5560 x 1640 x 12). Furthermore, in the overlapping case we have an additional 78,527 (overlapping) documents (288,625 - 210,158). For projection purposes if we assume that each overlapping document is an average document containing 141 terms, then the cost these additional documents will introduce to IIS is 127MB (78,527 x 141 (average number of terms per document) x 12), thus both (104 MB + 127MB) make a total of 231MB. In Table VIII, the difference between (actual) overlapping ICsIIS and FS is slightly less than this number and is equal to 222MB (about 4% less than our projection.).

It is possible to decrease the storage cost of inverted file structures by almost 50% by replacing the term weight information (8 bytes) by term document frequency (2 bytes) [Moffat, Zobel, 1996]. If we do that relative retrieval performances are expected to remain the same and the cost of individual posting list elements drops from 12 (4 + 8) bytes to 6 (4 + 2) bytes.

The detailed disk storage requirements for the important file structures of each strategy are shown in Table IX. The last two rows of the table show the storage overhead of storing the indexing terms to find the posting lists of the query terms.

Table IX. Storage requirements (in MB) for individual components

Storage Component	**Size**	**FS**	**ICISS**	**ICsISS**
IIS (inverted index for docs.)	338	*	*	
IIS with skip information	426			*
IC (Centroid length 250)	5		*	*
CM (Cluster Membership)	3		*	
Wordlist	5	*	*	*
Centroid word list (cent. size 250)	5		*	*

* Means that corresponding component is required.

The in-memory requirements of ICsIIS are similar to that of ICIIS. However, ICsIIS does not require cluster membership information to be kept in the memory, since it is blended into the posting lists, whereas ICISS does. Accordingly, the most demanding internal storage requirement for ICsIIS is for the so-called *accumulator array*, which is used to store the similarity of documents to the processed query. This requirement is clearly the same for all

three strategies described in this paper. From these discussions, we can conclude that ICsIIS is feasible in terms of memory and disk storage costs.

4.5 Effects of Centroid Generation Strategies

In the experiments, we investigated the impact of centroid length and centroid term weighting schemes on the effectiveness and efficiency of cluster based retrieval by using the *Qmedium* case as a representative. All the experiments are performed over clusters generated by both non-overlapping and overlapping versions of the C^3M algorithm. We generated four sets of centroids with fixed lengths 250, 500, 1000 and 2500. In an additional experiment, we set the centroid length of each cluster to the 10% of its unique terms. For each of these lengths, we applied three different centroid term weighting schemes: CW1, CW2, and CW3, where the weight of a centroid term is computed by the formula *term frequency* x *IDF*. In CW1, *term frequency* is taken as *1*, in CW2 and CW3 it is taken as the number of occurrence of the term in the cluster. In CW1 and CW2, *IDF* is taken as *ln(number of clusters/number of centroids including the term) + 1*, in CW3, it is taken as *ln(sum of occurrence numbers in the centroids/number of occurrence in the cluster) + 1*. All weights are normalized once they are assigned.

For all of these experiments, the effectiveness of CBR remains almost the same; whereas the efficiency slightly degrades as accessing inverted index elements for centroids requires more time with increasing centroid length. However, in all experiments, ICsIIS still outperforms ICISS in terms of query processing time. Also, in most of the experiments, ICsIIS achieved comparably well as FS, which is not influenced from the change of centroids.

4.6 Scalability Experiments

The scalability of C^3M, especially from an incremental clustering point of view, has been thoroughly studied in our previous work [Can, 1993; Can et al. 1995]. In this section we consider the scalability of our skip-based CBR strategy in terms of its efficiency, effectiveness, and storage structures. For obtaining the clusters, we use a naive implementation of C^3M based on ASCII files. In a PC environment, this unrefined implementation clusters the FT database in approximately 114 minutes.

For the scalability experiments we obtained two smaller versions of the FT database containing approximately one third and two thirds of the original collection. We refer to them as FT small (FTs) and FT medium (FTm). The characteristics of all FT databases are given in Table X (for easy reference the original FT database is also repeated in the same table). FTs and FTm, respectively, contain the first *69,507* and *138,669* documents of the original FT database. It may be noted in passing that the indexing clustering relationships are again

observed. For example, the clustering indexing relationship $n_C = n / x_d$ implies *989* and *1345* documents, respectively, for the FTs and FTm databases. The difference between actual numbers and projected numbers is less than *4%* as in the case of FT (see Section 4.1.1).

Table X. Characteristics of the FT Databases

Database	m No. of Documents	n No. of Terms	x_d Avg. No. of Distinct Terms/Doc.	n_C No. of Clusters	d_C Avg. No. of Docs./Clust.
FTs	69,507	144,080	145.7	955	73
FTm	138,669	191,112	142.1	1319	105
FT	210,158	229,748	140.6	1,640	128

In the scalability experiments, as a representative case, we only consider the non-overlapping clustering structure and use the *Qmedium* query set, which is the mid-way in terms of the query sizes we used. In the experiments we retrieve *10%* of the clusters (n_s= *0.1* x n_c), examine the top *10* documents (d_s= *10*) for performance measurement, and use centroids with 250 terms as in the previous experiments.

Scalability of Effectiveness

The experimental results in terms of single mean average precision value are reported here. Table XI shows that when we use the small database, FTs, the CBR effectiveness is about 16% lower than that of FS. In the case of FTm the performance of CBR in terms of effectiveness improves and it lags behind FS by only 1%. Finally, with the full and largest database, CBR outperforms FS by 4%. These observations confirm that our CBR methodology scales well with the database size and has the tendency of showing slightly better performance than that of FS with larger databases. This improvement of CBR effectiveness can be attributed to the refinement of cluster structures with increasing database size.

Table XI. Mean average precision values for retrieval strategies FS and CBR with different databases (n_s = *10% of* n_c, d_s =*10) and performance of CBR wrt FS*

Database	FS	CBR	CBR / FS
FTs	**0.285**	0.238	0.84
FTm	**0.335**	0.332	0.99
FT	0.314	**0.326**	1.04

Scalability of Efficiency

Table XII provides the average number of posting list elements processed for a query with each database. The values of the ICsIIS column show that this approach is much more efficient than the other two approaches. The last column of the table shows the savings provided by ICsIIS with respect to FS and ICIIS in terms of the posting list elements processed. For FTs, it provides 46% savings and savings increase with the increase of the database size and finally for FT the savings provided by ICsIIS are a substantial *67%*. As shown in Table XIII, these savings translate themselves to in-memory processing time

savings. The last column of the table shows that the efficiency of ICsIIS becomes more prevalent with increasing database size. This again shows that ICsIIS scales well with increasing database sizes. Note that, comparable efficiency results for overall query processing times (with I/O) are also observed in our experimental environment.

Table XII. Average number of "document" posting list elements processed by each retrieval strategy for each database and percentage savings provided by ICsIIS

Database	FS and ICIIS	ICsIIS	% ICsISS savings wrt FS and ICIIS
FTs	16,875	9,214	46
FTm	32,916	14,161	57
FT	49,415	16,342	67

Table XIII. Average in-memory processing time (sec.) per query for each retrieval strategy with each database and relative performance of ICsIIS with respect to FS

Database	FS	ICISS	ICsIIS	ICsIIS/FS
FTs	0.043	0.044	**0.022**	**0.51**
FTm	0.092	0.091	**0.045**	**0.49**
FT	0.141	0.143	**0.055**	**0.39**

Scalability of Storage and Indexing Structures

For the experimental databases FTs, FTm, and FT the requirements of the individual storage components are shown in Table XIV. As the numbers show, the overhead of the secondary storage structures (namely inverted index for centroids –IC–, cluster membership information –CM–, and the word lists used to find the posting lists associated with the query terms -the last two rows-) is negligible. For example, the storage cost of IC with respect to IIS is *2.6%*, *1.8%*, and *1.5%* for the databases FTs, FTm, and FT, respectively. As the size of the database increases, the relative cost of IC decreases, since the rate of increase in number of clusters is lower than that of documents. The size of the IC storage structure also indicates that in query processing the cost of selecting the best matching clusters is a small fraction of the query processing time. In terms of storage requirements, numbers are basically proportional to the sizes of the document vectors used for the creation of the index structures. As we increase the size of the database, the cost of skip-based IIS slightly decreases (from *0.30* to *0.26*) with respect to IIS. This is again due to the fact that the rate of increse in number of clusters is smaller than that of documents.

In terms of centroid (and IIS) generation, we have the following time observations respectively for FTs, FTm, and FT: *5*(*46*), *11*(*109*), and *20*(*182*) minutes. The time requirements of generating IIS and skip-based IIS are almost the same. Since these are one-time costs and our concern was the scalability of efficiency and effectiveness, we did not try to optimize our implementations for the generation of these storage structures.

Table XIV. Storage requirements (in MB) for individual components

Storage Component	**FTs**	**FTm**	**FT**
IIS (inverted index for docs.)	116	226	338
IIS with skip information (extra overhead wrt IIS)	151 (0.30)	286 (0.27)	426 (0.26)
IC (centroid length 250)	3	4	5
CM (Cluster Membership)	1	2	3
Wordlist	3	4	5
Centroid word list (cent. size 250)	3	4	5

4.7 Discussion of Results

From the experiments, we draw the following conclusions:

- In the non-overlapping clustering experiments, all three retrieval strategies of FS, ICISS and ICsIIS achieve similar effectiveness values. In the efficiency experiments, the ICsIIS strategy incorporated with a skip-based inverted index outperforms the other strategies in terms of in-memory operations and performs comparably well in terms of overall query processing times, i.e., including I/O, with effective OS file caching for centroid index.
- In the overlapping clustering experiments; the effectiveness values of ICISS and ICsIIS are slightly improved in particular experiments, but the efficiency results are not as good as the non-overlapping case due to the increasing access costs for both CBR strategies.
- The results are independent of the centroid lengths and weighting schemes, as the variations over these parameters do not significantly affect the presented results.
- Storage requirements in the disk and memory for ICIIS and ICsIIS are moderately higher than FS, and current compression techniques may further reduce these requirements. In ICsIIS, such a reduction has the potential of further improving the processing time, since by using our skipping approach the decompression time can be reduced significantly.
- The experiments show that our results are scalable: Effectiveness of CBR slightly increases and efficiency of ICsIIS can improve significantly with increasing database sizes.

5. Previous and Related Work

A good survey of clustering in information retrieval is provided in [Willett, 1988]. This work comes with an impressive reference list. The books by Salton [Salton, 1975, 1989], Salton and McGill [Salton, McGill, 1983] and van Rijsbergen [van Rijsbergen, 1979] also cover previous work on clustering in information retrieval. A new survey of clustering in various application areas can be found in [Jain et al., 1999]. A good discussion of algorithms for clustering data and cluster validation approaches is available in a beautiful concise book by Jain and Dubes [Jain, Dubes, 1988].

Our previous work on C^3M includes its concepts and effectiveness analysis [Can, Ozkarahan, 1990], and how it works in dynamic databases [Can, 1993; Can et al., 1995]. The CBR effectiveness in terms of precision for the INSPEC database is reported in [Can & Ozkarahan, 1990]. The study shows that C^3M is 15.1 to 63.5 (with an average of 47.5) percent better than four other clustering algorithms [El-Hamdouchi, Willett, 1989] in CBR. The same study also shows that the IR effectiveness of the algorithm is comparable with a demanding (in terms of CPU time and main memory) complete link clustering method that is known to have good retrieval performance [Voorhees, 1986a; Voorhees, 1986b]. The experiments also show that the CBR using C^3M is slightly less effective (1.0 percent to 6.9 percent) than FS. The experimental observations reported in [Can et al., 1995] show that the incremental version of C^3M is cost effective and can be used for many increments of various sizes.

C^3M and its concepts have also attracted the attention of other researchers in various application areas, such as chemical information systems [Willett, et al., 1986], clustering tendency testing [Elhamdouchi, Willet, 1987], automatic hypertext structure generation [Kellogg, Madhan, 1996], and search output clustering [Kural et al., 2001].

Most clustering research in IR is related to cluster search effectiveness [Griffiths et al., 1986; Willett, 1988; Burgin 1995; Shaw et al. 1997; Schütze, Craig 1997]. The research on efficiency aspects of cluster searches is limited. For example, the works presented in [Can, 1994, Voorhees, 1986b] considers storage, CPU, and I/O efficiency in the same simulated environment. The Salton-McGill book [Salton, McGill, 1983] approaches to the efficiency problem in terms of page faults during information retrieval.

The studies reported in [Hearst, Pedersen, 1996; Silverstein, Pedersen, 1997] provide experiments using databases larger than our collection; however, in their evaluations they assume that user picks the optimal cluster or try to generate refined cluster via scatter/gather browsing paradigm based on an existing global clustering structure. In contrast to these approaches, in our work all decisions are made automatically using similarity measures based on a pre-existing clustering structure with no user interaction or any other assumption.

In FS only query term posting lists are accessed from the disk medium. As a result, the efficiency of FS decreases with increasing query length, since for each query term another posting list must be processed. It is possible to employ a partial evaluation (or pruning) strategy that skips some of the query terms to improve search efficiency with similar search effectiveness [Buckley, Lewit, 1985; Brown, 1995; Persin, 1994; Moffat, Zobel, 1996]. However, as it is stated before such an approach is beyond the scope of this study and inverted index search optimization in CBR (i.e., in ICIIS and ICsIIS) is an interesting research possibility by itself, which can be further incorporated to our work.

6. Conclusions and Future Work

Our CBR implementation method employs a storage structure that blends in the cluster membership information with the inverted file posting lists using the concept of *skips*. In the skip approach, posting lists contain the cluster membership information in addition to traditional term weighting information. During CBR, skip pointers embedded in posting lists provide the information to skip unnecessary (non-best matching) cluster members. The indexing structure of the skip approach can be used both for FS and CBR. Our skip-based CBR significantly improves the efficiency of query processing and this improvement is especially due to in memory similarity calculations. As web search engines often need to traverse very long posting lists in memory, our skip-based CBR would improve the efficiency of web search engines that may employ clustering. Our results are significant in the sense that the efficiency and effectiveness of CBR have been analyzed at this level for the first time for an existing global clustering structure (note that the clustering structure is static at the time of query processing; however it can be updated in an incremental manner at other times [Can 1993; Can et al. 1995]).

We show that for large databases CBR can achieve a time efficiency and effectiveness comparable with FS. The storage requirement for CBR is modestly higher than the ordinary inverted index file. The experiments show that our results are scalable: Effectiveness of CBR slightly increases and efficiency of ICsIIS can improve significantly with increasing database sizes.

There are several promising future research directions:

1. In the experiments it is observed that CBR and FS do not always return exactly the same set of relevant documents even when they achieve the same precision levels; therefore, our results are also important in terms of data fusion or mixing the results of FS and CBR [Griffiths et al. 1986; Lee, 1997]. Our skip-based storage structure is especially suitable for an unusual fusion method, which is a hybrid of FS and CBR. For instance, for *important* query terms with relatively high weights, we may *turn-off* skipping, to retrieve some of the documents that are not in the best matching clusters but still qualify to be in the top 10 (20) documents. We are currently studying other possible heuristics that may allow combining the best possible results from FS and CBR, with the least additional overhead.
2. It would be interesting to study the update of the skip-based inverted index structures in a dynamic retrieval environment with new and deleted old documents.
3. Compression of the posting lists and its effect on the system efficiency both in terms of retrieval time and disk space is another promising research direction. There is every reason to expect that compression will have positive effects on performance,

since with compression a similar skip approach gives good results [Moffat, Zobel, 1996].

4. Another research direction is definition of document vectors with different levels of indexing exhaustivity [Burgin, 1995] or by latent semantic indexing (LSI) and measuring the system performance [Lee, 1997; Schütze, Silverstein, 1997].
5. Indexing of documents at a lower level, such as paragraphs or sentences, looks promising from CBR's point of view. Since in such an environment FS inverted indexes could be extremely long, our optimization with skip concept combined with CBR may provide an important efficiency leap during query processing.
6. For the calculation of the similarity values instead of an *accumulator array* dynamic data structures can be used for memory efficiency [Witten, 1994]. A partial query evaluation or pruning strategy and its effectiveness and efficiency should also be investigated [Brown, 1995; Moffat, Zobel, 1996; Persin 1994]. Our scalability experiments (Table XIII) and the results reported in [Moffat, Zobel, 1996] imply that when our ICsIIS approach is combined with the *restricted accumulators* (*quit* and *continue* methods of Moffat and Zobel), it can further improve the efficiency performance.

References

R. Baeza-Yates, R. Riberio-Neto, *Modern Information Retrieval*, Addison Wesley, Reading, MA, 1999.

E. W. Brown, Fast evaluation of structured queries for information retrieval, In *Proceedings of the 18th Annual International ACM-SIGIR Conference*, ACM, New York, 1995, pp. 30-38.

R. Burgin, The retrieval effectiveness of five clustering algorithm as a function of indexing exhaustivity, *Journal of the American Society for Information Science* 40 (1995) 562-572.

C. Buckley, A. F. Lewit, Optimization of inverted vector searches, In *Proceedings of the 8th Annual International ACM-SIGIR Conference*, ACM, New York, 1985, pp. 97-110.

F. Can, On the efficiency of best-match cluster searches, *Information Processing and Management* 30 (1994) 343-361.

F. Can, Incremental clustering for dynamic information processing, *ACM Transactions on Information Systems* 11 (1993) 143-164.

F. Can, E. A. Fox, C. D. Snavely, R. K. France, Incremental clustering for very large document databases: initial Marian experience, *Information Sciences* 84 (1995) 101-114.

F. Can, E. A. Ozkarahan, Concepts and effectiveness of the cover-coefficient-based clustering methodology for text databases, *ACM Transactions on Database Systems* 15 (1990) 483-517.

F. Can, E. A. Ozkarahan, Two partitioning type clustering algorithms, *Journal of the American Society for Information Science* 35 (1984) 268-276.

A. Griffiths, H. C. Luckhurst, P. Willet, Using interdocument similarity information in document retrieval systems, *Journal of the American Society for Information Science* 37 (1986) 3-11.

A. El-Hamdouchi, P. Willett, Techniques for the measurement of clustering tendency in document retrieval systems *Journal of Information Science.* 13 (1987) 361-365.

M. A. Hearst, J. O. Pedersen, Reexamining the cluster hypothesis: Scatter/gather on retrieval results, In *Proceedings of the 19th ACM-SIGIR Conference*, ACM Press, 1996, pp. 74-81.

A. K. Jain, R. C. Dubes, *Algorithms for Clustering Data*, Prentice Hall, Upper Saddle River, NJ, 1988.

A. K. Jain, M. N. Murty, P. J. Flynn, Clustering algorithms: A review, *ACM Computing Survey* 31 (1999) 264-323.

N. Jardine, C. J. van Rijsbergen, The use of hierarchical clustering in in information retrieval, *Information Storage and Retrieval* 7 (1971) 217-240.

R. B. Kellogg, M. Subhas, Text to hypertext: Can clustering solve the problem in digital libraries, In *Proceedings of the 1st ACM International conference on Digital Libraries*, ACM, New York, 1996, pp. 144-150.

S. Kocberber, F. Can, J. M. Patton, Optimization of signature file parameters for databases with varying record lengths, *The Computer Journal* 42 (1999) 11-23.

Y. Kural, S. Robertson, S. Jones, Deciphering cluster representations, *Information Processing and Management* 37 (2001) 593-601.

J. H. Lee, Analyses of multiple evidence combination. In *Proceedings of the 20th ACM-SIGIR Conference*, ACM Press, 1997, pp. 267-276.

A. Moffat, J. Zobel, Self-indexing inverted files for fast text retrieval, *ACM Transactions on Information Systems* 14 (1996) 349-379.

M. Persin, Document filtering for fast ranking. In *Proceedings of the 17th ACM-SIGIR Conference*, ACM Press, 1994, pp. 339-348.

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

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

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

G. Salton, C. Buckley, Term-weighting approaches in automatic text retrieval, *Information Processing and Management* 24 (1988) 513-523.

H. Schütze, C. Silverstein, Projections for efficient document clustering, In *Proceedings of the 20th ACM-SIGIR Conference*, ACM Press, 1997, pp. 74-81.

W. M. Shaw Jr., R. Burgin, P. Howell, Performance standards and evaluations in IR test collections: Cluster-based retrieval models, *Information Processing and Management* 33 (1997) 1-14.

C. Silverstein, J. O. Pedersen, Almost-constant-time clustering of arbitrary corpus subsets, In *Proceedings of the 20th ACM-SIGIR Conference*, ACM Press, 1997, pp. 60-66.

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

E. M. Voorhees, The effectiveness and efficiency of agglomerative hierarchical clustering in document retrieval. Ph.D. Thesis, Cornell Univ., Ithaca, NY, 1986a.

E. M. Voorhees, The efficiency of inverted index and cluster searches In *Proceedings of the 9th Annual International ACM-SIGIR Conference*, ACM, New York, 1986b, pp. 164-174.

P. Willett, Recent trends in hierarchical document clustering: A critical review, *Information Processing and Management* 24 (1988) 577-597.

P. Willett, V. Wınterman, D. Bawden, Implementatıon of non-hierarchıcal cluster-analysis methods in chemical ınformation-systems - selection of compounds for biologıcal testing and clustering of substructure search output, *Journal of Chemical Information And Computer Sciences* 26 (1986) 109-118.

I. H. Witten, A. Moffat, T. C. Bell, *Managing Gigabytes Compressing and Indexing Documents and Images.* Van Nostrand Reinhold, New York, 1994.

S. B. Yao, Approximating block accesses in database organizations, *Comm. of the ACM* 20 (1977) 260-261.

C. T. Yu, W. Meng, *Principles of Database Query Processing for Advanced Applications*. San Francisco, CA, Morgan Kaufmann, 1998.