Incremental Cluster-Based Retrieval Using Compressed Cluster-Skipping Inverted Files

ISMAIL SENGOR ALTINGOVDE, ENGIN DEMIR, FAZLI CAN,
and ÖZGÜR ULUSOY
Bilkent University

We propose a unique cluster-based retrieval (CBR) strategy using a new cluster-skipping inverted file for improving query processing efficiency. The new inverted file incorporates cluster membership and centroid information along with the usual document information into a single structure. In our incremental-CBR strategy, during query evaluation, both best(-matching) clusters and the best(-matching) documents of such clusters are computed together with a single posting-list access per query term. As we switch from term to term, the best clusters are recomputed and can dynamically change. During query-document matching, only relevant portions of the posting lists corresponding to the best clusters are considered and the rest are skipped. The proposed approach is essentially tailored for environments where inverted files are compressed, and provides substantial efficiency improvement while yielding comparable, or sometimes better, effectiveness figures. Our experiments with various collections show that the incremental-CBR strategy using a compressed cluster-skipping inverted file significantly improves CPU time efficiency, regardless of query length. The new compressed inverted file imposes an acceptable storage overhead in comparison to a typical inverted file. We also show that our approach scales well with the collection size.

Categories and Subject Descriptors: E.4 [**Data**]: Coding and Information Theory—*Data compaction and compression*; H.3.2 [**Information Storage and Retrieval**]: Information Storage—*File organization*; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Clustering, search process*

General Terms: Experimentation, Measurement, Performance

Additional Key Words and Phrases: Best match, cluster-based retrieval (CBR), cluster-skipping inverted index structure (CS-IIS), full search (FS), index compression, inverted index structure (IIS), query processing

ACM Reference Format:

Altingovde, I. S., Demir, E., Can, F., and Ulusoy, Ö. 2008. Incremental cluster-based retrieval using compressed cluster-skipping inverted files. ACM Trans. Inform. Syst. 26, 3, Article 15 (June 2008), 36 pages. DOI = 10.1145/1361684.1361688 http://doi.acm.org/10.1145/ 1361684.1361688.

This research is supported by The Scientific and Technical Research Council of Turkey (TÜBITAK) under the grant no. 105E024.

Authors' addresses: I. S. Altingovde, E. Demir, F. Can, and Ö. Ulusoy, Computer Engineering Department, Bilkent University, Ankara, 06800, Turkey; email: {ismaila, endemir, canf, oulusoy}@cs.bilkent.edu.tr.

1. INTRODUCTION

In an information retrieval (IR) system the ranking-queries, or Web-like queries, are based on a list of terms that describe the user's information need. Search engines provide a ranked document list according to potential relevance of documents to user queries. In ranking-queries, each document is assigned a matching score according to its similarity to the query using the vector space model [Salton 1989]. In this model, the documents in the collection and queries are represented by vectors whose dimensions correspond to terms in the vocabulary of the collection. The value of a vector entry can be determined by one of the several term weighting methods proposed in the literature [Salton and Buckley 1988]. During query evaluation, query vectors are matched with document vectors by using a similarity function. The documents in the collection are then ranked in decreasing order of their similarity to the query and the ones with highest scores are returned. Note that Web search engines exploit the hyperlink structure of the Web or the popularity of a page for improved results [Brin and Page 1998; Long and Suel 2003].

However, exploiting the fact that document vectors are usually very sparse, an inverted index file can be employed instead of full vector comparison during the ranking-query evaluation. Using an inverted index, the similarities of those documents that have at least one term in common with the query are computed. In this article, a ranking-query evaluation with an inverted index is referred to as full search (FS). Many state-of-the-art large-scale IR systems such as Web search engines employ inverted files and highly optimized strategies for ranking-query evaluation [Zobel and Moffat 2006].

An alternative method of document retrieval is first clustering the documents in the collection into groups according to their similarity to each other. Clusters are represented with centroids which can include all or some of the terms that appear in the cluster members. During query processing, only those clusters that are most similar to the query are considered for further comparisons with cluster members, that is, documents. This strategy, so-called cluster-based retrieval (CBR), is intended to improve both the efficiency and effectiveness of document retrieval systems [Jardin and Van Rijsbergen 1971; Salton 1975; Salton and McGill 1983; Voorhees 1986b]. CBR can improve efficiency, as the query-document matches are computed for only those documents that are in clusters most similar to the query. Furthermore, it may enhance effectiveness, according to the well-known cluster hypothesis [van Rijsbergen 1979; Voorhees 1985]. Note that the resulting ranking returned by CBR can be different from that of FS, as the former considers only those documents in promising clusters.

Surprisingly, despite these premises of CBR for improving effectiveness and efficiency, the information retrieval community has witnessed contradictory results in terms of both of these aspects in the last few decades [Salton 1989; Voorhees 1986b; Liu and Croft 2004]. This inconsistency has reduced interest in CBR and its consideration as an alternative retrieval method to full search. On the other hand, the growth of the Web as an enormous digital repository of every kind of media, and essentially of text, also creates new opportunities for the use of clustering and CBR. For example, Web directories (e.g., DMOZ, Yahoo, etc.), a major competitor of search engines, allow users to browse through the categories and assign a query on a particular one. This is a kind of CBR, except that clusters are browsed manually. Furthermore, there exist several large-scale text repositories that are available on Web or on proprietary networks, again with manual and/or automatic classification/clustering of the content. Clearly, CBR as a model of information retrieval perfectly fits the requirements of such environments, provided that the suspected obstacles to its effectiveness and efficiency are remedied. A recent attempt at addressing the effectiveness front is by Liu and Croft [2004], which shows that by using language models, CBR effectiveness can be significantly better than assumed in the literature. The efficiency of CBR is investigated in this article.

For any given IR system involving document clusters (or categories), created either automatically or manually, for legacy data or Web documents and in a flat or hierarchical structure, the best-match CBR strategy has two stages: (i) best(-matching) cluster selection, where those clusters most similar to the submitted query are determined by using cluster centroids; and (ii) best(-matching) document selection, wherein the documents from these best-matching clusters are matched with the query to obtain the final query result. In the early days of IR, once best clusters are obtained, it was presumed a reasonable strategy to compare the query with the document vectors of the members of those clusters (exhaustive search). This may be a valid and efficient strategy if the clusters are rather small and queries rather long. However, state-of-the-art applications for CBR, such as Web directories or digital libraries, involve collections with large numbers of documents with respect to the number of clusters (i.e., categories) and attempt to respond to a very high load of typically short queries. Indeed, the inefficiency of the exhaustive strategy has been long recognized [Salton 1989; Voorhees 1986b]. As a remedy, the use of inverted index files for both stages of CBR (i.e., comparison with centroids and documents) has been proposed [Can 1994] (see Section 2.2.2 for a more detailed discussion). More specifically, once the best clusters are obtained, a full search is conducted over the entire collection to find those documents that have nonzero similarity to the query, that is, the candidates to be the best documents. Next, from among these documents, only those from the best clusters are filtered to be presented to the user. This is a practical approach that is also applied in current-day systems [Cacheda et al. 2003; Cacheda and Baeza-Yates 2004]. In this article, we refer to this strategy as typical CBR.

However, typical CBR still involves some significant redundancy. At the best-document selection stage, the inverted index is used to find “all” documents that have nonzero similarity to the query (note this is nothing but FS). Since only documents from the best clusters are returned, the computations (decoding postings, computing partial similarities, updating accumulators, inserting into and extracting from the heap for the final output, etc.) for the eliminated documents are all wasted. Furthermore, there is the cost of computing best clusters. If the index files are kept on disk (a relaxable assumption, considering the advances in the hardware, as we discuss later), accessing these structures requires two direct (random) disk accesses per query term: one for the centroid and another for document posting lists. These issues imply that typical CBR as defined here cannot be a competitor of FS in terms of efficiency, as it already involves the cost of FS in addition to the aforementioned costs specific to the best-cluster selection stage.

Note the several recent approaches to optimize the basic FS strategy by applying dynamic pruning techniques [Persin 1994; Persin et al. 1996; Anh and Moffat 2006, 2005b, 2001; Lester et al. 2005]. These may equivalently improve the second stage of typical CBR as well. However, none of these approaches exploits specific information based on clustering, and thus the additional costs explained in the previous paragraph still remain. In this article, we attempt to optimize both stages of typical CBR so that almost no redundant work is done. On top of this, it still may be possible to apply other optimizations, which we briefly discuss later.

In this article, our goal is to design a CBR strategy that can overcome the efficiency weaknesses of typical CBR and be as efficient as FS, while providing comparable effectiveness to FS and typical CBR. We introduce a cluster-skipping inverted index structure (CS-IIS) and, based on this structure, a unique cluster-based retrieval strategy. In the CS-IIS file, cluster membership- and within-cluster term frequency information are embedded into the inverted index, extending an approach in earlier works [Can et al. 2004; Altingovde et al. 2007] which were inspired from Moffat and Zobel [1996]. Different from previous studies, centroids are now stored in the original term posting-lists and used for document matching. This enhanced inverted file eliminates the need for accessing separate posting lists for centroid terms. In the new CBR method, the computations required for selecting the best-matching clusters and as well as for the best-matching documents of such clusters are performed together in incremental and interleaved fashion. The query terms are processed in a discrete manner in nonincreasing term-weight order. In other words, we envision a term-at-a-time query processing mode in this article, whereas another highly efficient alternative, document-at-a-time, is out of scope [Anh and Moffat 2006]. As we switch from the current query term to the next, the set of best clusters is recomputed and can dynamically change. In the document matching stage of CBR, only those portions of the current query-term posting list that correspond to the latest best-matching cluster set are considered. The rest are skipped, hence not involved in document matching. During document ranking, only those members of the most recent best-matching clusters with a nonzero similarity to the query are considered.

In the literature, it is observed that the size of an inverted index file can be very large [Witten et al. 1994]. As a remedy, several efficient compression techniques are proposed that significantly reduce the file size. In this study, without loss of generality, we concentrate on the IR strategies with compression where the performance gains of our approach become more emphasized. Indeed, our incremental-CBR strategy with the new inverted file is tailored to be most beneficial in such a compressed environment. In other words, skipping irrelevant portions of the posting lists during query processing eliminates the substantial decompression overhead (as in Moffat and Zobel [1996]) and provides further improvement in efficiency. In compression, we exploit the use of multiple posting-list compression parameters and reassign document ids of individual cluster-members to increase the compression rate, as recently proposed in the literature [Blanford and Blelloch 2002; Silvestri et al. 2004].

The proposed approach promises significant efficiency improvements: If memory is scarce (e.g., for digital libraries and proprietary organizations) and index files have to be kept on disk, the incremental-CBR algorithm with CS-IIS allows the queries to be processed by only one direct disk access per query term. Furthermore, even if the centroid and/or document index is stored in memory, which is probable with the recent advances in hardware (see Strohman and Croft [2007] as an example), the CS-IIS saves decoding and processing those document postings that are not from best clusters, a nontrivial cost. We show that the most important overhead of CS-IIS, namely, longer posting lists, is reduced to an affordable overhead by our compression heuristics and that even with a moderate disk, the gains in efficiency can compensate for slightly longer disk-transfer times (given that number of clusters tends to be much smaller than that of documents).

Our comparative efficiency experiments cover various query lengths and both storage size- and execution-time issues in a compressed environment. The results, even with lengthy queries, demonstrate the robustness of our approach. We show that our approach scales well with the collection size. In the experiments, we use multiple query sets and three datasets of sizes 564MB, 3GB, and 27GB, corresponding to 210,158 and 1,033,461, and 4,293,638 documents, respectively.

1.1 Motivation

The huge amount of digitally available text implies new opportunities for the use of clustering and CBR. For instance, hierarchic taxonomies, in the form of Web directories or in digital libraries, allow users to browse through categories or to issue queries that are restricted to a certain subset of these categories [Cacheda et al. 2003; Cacheda and Baeza-Yates 2004], in addition to the usual keyword searches over the entire collection. Such directories, though first created manually (e.g., DMOZ), have potential for further growth by using machine learning methods. Also, clustering can be employed for less constrained collections, either in association with or independently from supervised categorization. All these environments call for efficient methods for processing queries restricted to a certain cluster(s), which may be determined automatically (as we assume here) or browsed by the user. This is clearly a sort of CBR, as we mean in this article.

CBR may also prove beneficial for presenting query results. Assuming results are presented based on their clusters, it is possible for the user to browse a cluster and thereby discover some other similar and potentially relevant documents which do not capture any of the query words, and which are unreachable otherwise. Furthermore, the user can pose a refined query in a cluster that (s)he presumes relevant, which may improve user satisfaction and system efficiency. Again, such clustered environments would require efficient methods of conducting CBR. The existence of such methods would accelerate the motivation for clustering and/or categorizing the content for user access.

In this article, we envision that CBR is a worthwhile strategy in certain domains such as those described earlier, provided that it can be competitive with FS in terms of both effectiveness and efficiency. Given the recent promising findings on effectiveness [Liu and Croft 2004], we focus on the latter and thus aim to show that efficient CBR is an attainable goal as well.

1.2 Contributions

The contributions of this study consist of the following.

—*Introducing a Pioneering CBR Strategy*. We introduce an original CBR method using a new cluster-skipping inverted index structure and refer to it as incremental CBR. The proposed strategy interleaves query-cluster- and query-document matching stages of best-match CBR for the first time in the literature.

—*Embedding the Centroid Information in Document Inverted Indexes*. For memory-scarce environments (e.g., private networks, digital libraries, etc.) where the index files should be kept on disk, we eliminate the disk accesses needed for centroid inverted index posting-lists by embedding the centroid information in document posting-lists. This embedded information enables best-cluster selection by only accessing the document inverted index. In this way, during query processing, each query term requires only one direct disk access rather than separate disk accesses for centroid- and document posting-lists. The new data structure, cluster-skipping IIS (CS-IIS), is inspired from Can et al. [2004] and Moffat and Zobel [1996], but enriched as discussed later in the article.

—*Outperforming Full Search (FS) Efficiency*. We show that for large datasets, incremental CBR outperforms FS (as well as typical CBR, which actually involves FS during its best-document selection stage) in efficiency while yielding comparable (or sometimes better) effectiveness figures. We also show the efficiency of our approach to scale well with collection size. The proposed approach is also superior to the FS version that employs the "continue" pruning strategy accompanied with a skipping IIS, as described in Moffat and Zobel [1996].

—*Adapting the Compression Concepts to a CBR Environment*. We adapt multiple posting-list compression parameters and specify a cluster-based document id reassignment technique that best fits the features of CS-IIS.

—*CBR Experiments Using a Realistic Corpus Size with No User Behavior Assumption Performing*. We use the largest corpora reported in the CBR literature, assume no user interaction, and perform all decisions in an automatic manner. Only a few studies on CBR use collections as large as ours (e.g., Can et al. [2004], Liu and Croft [2004]).

The rest of the article is organized as follows. We start with reviewing the two traditional IR strategies, FS and typical CBR, which serve as the baseline cases for comparison with our incremental-CBR strategy. In Section 3, we introduce the incremental-CBR strategy using the cluster-skipping inverted index structure (CS-IIS). In Section 4, we discuss compression of the CS-IIS, with an emphasis on the benefits of document id reassignment in our framework. In Section 5, we describe the experimental environment and in Section 6, the proposed strategy is extensively evaluated and compared to an efficient FS implementation based on dynamic pruning and skips [Moffat and Zobel 1996]. Related work in the literature is reviewed in Section 7. Finally, we conclude and provide future research pointers in Section 8.

2. TRADITIONAL STRATEGIES FOR IR

In the following, we first review the two basic IR strategies, namely FS and typical CBR, as well as their implementations employing an IIS for ranking-queries. Finally, we briefly discuss compression techniques for the inverted files.

2.1 Full Search Using Inverted Index Structure

In an IR system, typically two basic types of queries are provided: Boolean and ranking-queries. In the former case, query terms are logically connected by the operators AND, OR, and NOT and those documents that make this logical expression true (i.e., satisfy the query) are retrieved. In ranking-queries, each document is assigned a matching score according to its similarity to the query, using the vector space model [Salton and McGill 1983]. In this work, we concentrate on the ranking queries, which are more frequently used in Web search engines (such as Google) and IR systems. However, our approach as proposed in this article can be applicable to Boolean queries, as well.

In the vector space model, the documents of a collection are represented by vectors. For a document collection including T distinct terms, each document is represented by a T-dimensional vector. For those terms that do not appear in the document, the corresponding vector entries are zero. On the other hand, the entries for those terms that appear in the document can be determined by one of several term weighting methods described in the literature [Salton and Buckley 1988]. The goal of these methods is to assign higher weights to those terms that can potentially discern one document among others, and vice versa. In this study, the document-term weights are assigned using the *term frequency (tf)* $\times$ *inverse document frequency (idf)* formulation. While computing the weight of term t in document d, denoted as $w_{d,t}$, term frequency is computed as the number of occurrences of t in d, and *idf* is *ln(number of all documents / number of documents containing t)* $+ 1$. During query processing, a term's weight is computed by using the *tf-idf* formula and then normalized by using the document lengths. Document lengths are computed with the formula

$$\sqrt{\sum_{t=1}^{T}(w_{d,t})^2}. \quad \text{(see Witten et al. [1994]) for further details).}$$

The term weights for query terms ($w_{q,t}$) are calculated in a similar fashion to document-term weights. In this case, for computing the term frequency component, we use an augmented normalized frequency formula defined as *(0.5 +* *0.5* x *tf / max-tf)*. Here *max-tf* denotes the maximum number of times that any term appears in the query vector. The *idf* component is obtained in exactly the same manner as 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 a T-dimensional vector space, the query-document matching is performed using the following formula [Salton and McGill 1983].

$$\mathit{similarity}(q, d) = \sum_{t=1}^{T} w_{q,t} \times w_{d,t}$$

It is possible to evaluate ranking-queries very efficiently by using an inverted index of document vectors. In this case, the query vector is not matched against all document vectors (most of which would probably yield no similarity at all), but only to those that have at least one term in common with the query. An inverted file has a header part, including a list of terms in the collection, and pointers to the posting lists for each term. Along with each term, f_t, namely the number of documents in which this term appears, is kept. A posting list for a term consists of those documents that include the term and is usually an ordered list of <document id d, within-document term frequency $f_{d,t}$> pairs (see Zobel and Moffat [2006] for other organizations).

During ranking-query evaluation, an accumulator array with as many entries as the collection size is kept in memory (note that variations are possible [Harman 1992]). The weighted query vector is constructed as described before. For each term t in the query vector q, a *direct access* is made to the disk to locate t's posting list by using the pointer stored in the IIS header. Once located, the posting list associated with this term t is read sequentially (assuming it is stored on contiguous disk blocks) and brought to main memory. For each document d in the posting list, first $w_{d,t}$ is computed by using the *tf-idf* formula. Note that the *tf* component corresponds to the $f_{d,t}$ values that are stored, along with the document ids, in the posting lists. The *idf* component can be easily computed using the term frequency f_t, stored in the IIS header. Next, using a similarity function, the partial similarity of query to document is computed (i.e., $w_{d,t} \cdot w_{q,t}$ for the cosine function [Witten et al. 1994]) for this particular term, and the resulting value is added to the accumulator entry for this document. After all query terms are processed in the same manner, the entries of the accumulator are normalized, that is, divided by the precomputed document lengths. Finally, the accumulators (documents) are sorted in descending similarity order and returned as the query output. If only the top-k documents are required and k is much smaller than the collection size (which is the common case, as in the Web), using a heap data structure significantly reduces the query processing time. Details of ranking-query processing are discussed extensively in Cambazoglu [2006], Cambazoglu and Aykanat [2006], and Witten et al. [1994].

In this article, a ranking-query evaluation as described in the preceding discussion is referred to as full search. It is "full" in the sense that it returns exactly the same results as the sequential collection scan and uses all terms in the documents (except for stop-words, as we mention in the experimental setup).

2.2 Cluster-Based Retrieval (CBR)

Document clustering can produce a hierarchic or flat structure (see the beautiful book by Jain and Dubes [1988] for a review of clustering algorithms). Most research in the literature focuses on the hierarchic methods (see, e.g., Willett [1988]). Cluster-based retrieval is intended to improve both the efficiency and effectiveness of retrieval systems [Jardin and van Rijsbergen 1971; Salton 1975; Voorhees 1986b]. In this study, we focus on the partitioning clustering- and best-match CBR. In the following subsections, we first review several centroid construction techniques for representing clusters and then we discuss typical CBR with IIS.

2.2.1 *Cluster Centroids and Centroid Weighting.* A classical problem of cluster-based retrieval is selecting the terms in the cluster centroids and determining the maximum centroid length as well as centroid-term weights. Murray [1972] states that the effectiveness of retrieval does not increase linearly with maximum centroid length. Thus, in the literature, typically a limited number of terms selected by various methods are used as cluster centroids. For instance, in the hierarchical clustering experiments described by Voorhees [1986a, 1986b], the sum of within-document frequency of each term in a cluster is computed and the terms are sorted by decreasing frequency. Next, top-k terms are selected to comprise the cluster centroid, where an appropriate value of k is experimentally determined [Voorhees 1986a]. Note that, based on Murray's centroid definition [1972], Voorhees attempted to find those shortest centroid vectors that cause minimal deterioration of effectiveness. However, the results reported in that work show variability in drawing a conclusion on the relationship between centroid length and effectiveness for several hierarchical CBR techniques. In a recent work, several methods for centroid generation have been reviewed and it is concluded that the need for an extensive investigation of centroid influence on CBR effectiveness still exists [Tombros 2002].

In the rest of this article, we assume the use of three centroid-term weighting schemes: CW1, CW2, and CW3; in all of them the weight of a centroid term is computed by the formula $tf \times idf$. In CW1, while *tf* is taken as 1, while in CW2 and CW3 it is taken as the number of occurrences of a term in the cluster, that is, as the *within-cluster term frequency.* In CW1 and CW2, *idf* is taken as *ln(number of clusters / number of centroids including the term)* + 1, and in CW3 it is taken as *ln(sum of occurrence numbers in the centroids / number of occurrence in the cluster)* + 1. During the best-cluster selection stage of query processing, weights are normalized by using the precomputed cluster lengths. In Table I, the three centroid-term weighting schemes are summarized.

2.2.2 *Typical CBR Using Inverted Index Structure.* In partitioning clustering, a flat clustering of the documents is produced and the search is typically achieved by the best-match strategy. The best-match CBR search strategy has two stages: (i) selection of n_s, the number of best-matching clusters using centroids; and (ii) selection of d_s, the number of best-matching documents of the selected best-matching clusters. For item (i) we have two file structure possibilities: centroid vectors, and an IIS of centroids. For item (ii) we again have

Table I. Term Weighting Schemes for Centroids

Weighting Scheme	Term Frequency (*tf*)	Inverse Document Frequency (*idf*)
CW1	1	$\ln \frac{\text{number of clusters}}{\text{number of centroids including the term}} + 1$
CW2	within-cluster term frequency	$\ln \frac{\text{number of clusters}}{\text{number of centroids including the term}} + 1$
CW3	within-cluster term frequency	$\ln \frac{\text{sum of occurence numbers in the clusters}}{\text{number of occurence in the cluster}} + 1$

two possibilities: document vectors, and an IIS of all documents. One remaining possibility for (ii), namely, a separate inverted index for the members of each cluster, is not considered due to its excessive cost in terms of disk accesses (for a query with k number of terms, it would involve k direct disk accesses for each selected cluster) and maintenance overheads. Hence, possible combinations of (i) and (ii) determine four different implementation alternatives.

In Can [1994] the efficiency of the aforesaid alternatives is measured in terms of CPU time, disk accesses, and storage requirements in the simulated environment defined in Voorhees [1986b]. It is observed that the version employing an IIS for both centroids and documents (separately) is significantly better than the others. Notice that the query processing in this case is quite similar to the ranking-query evaluation for FS discussed in Section 2.1 and repeats that procedure twice, using centroid IIS and document IIS, respectively. A final stage is also required for filtering documents retrieved by the second stage (i.e., FS using the document index) but not belonging to the best clusters. A similar approach is typically used for processing queries that are restricted to certain categories on Web directories (with the only distinction being that best cluster(s) are explicitly specified by the user instead of by an automatic computation) [Cacheda et al. 2003; Cacheda and Baeza-Yates 2004]. Throughout the article, we consider this particular implementation as the baseline best-match CBR method and refer to it as *typical CBR*.

In Figure 1, we illustrate the centroid- and document IIS files for this strategy; the example provided in the figure is for a document-by-term D matrix with three clusters C_1, C_2, and C_3. In the D matrix, rows and columns respectively indicate documents and terms. The matrix shows that document 2 (d_2) contains term 1 (t_1) once and t_2 three times. We assume for simplicity that all terms appearing in the member documents of a cluster are used in the centroid and that the centroid inverted index is created accordingly. For instance, term t_1 appears in two documents, d_1 and d_2, once in each. Since both documents are in C_1, the posting element for C_1 in the list of t_1 stores the value 2 as the within-cluster term frequency.

In Can [1994], it is further stated that typical CBR is inferior to FS in terms of query evaluation efficiency. This is an expected result, as the best-document selection stage of typical CBR is actually nothing but a full search on the entire collection. Furthermore, selecting the best clusters as well as the final result filtering would also incur additional costs. In Altingovde et al. [2006], efficiency tradeoffs for typical CBR are discussed, but those arguments are essentially

$$D = \begin{array}{c|cccccc|c} & t_1 & t_2 & t_3 & t_4 & t_5 & t_6 & \\ \hline & 1 & 1 & 1 & 0 & 0 & 1 & d_1 \\ & 1 & 3 & 1 & 1 & 0 & 0 & d_2 \\ & 0 & 0 & 0 & 3 & 2 & 1 & d_3 \\ & 0 & 0 & 1 & 7 & 0 & 0 & d_4 \\ & 0 & 0 & 0 & 0 & 1 & 5 & d_5 \\ & 0 & 0 & 0 & 0 & 1 & 4 & d_6 \\ & 0 & 0 & 1 & 0 & 1 & 1 & d_7 \end{array}$$

$C_1 = \{d_1, d_2\}$
$C_2 = \{d_3, d_4\}$
$C_3 = \{d_5, d_6, d_7\}$

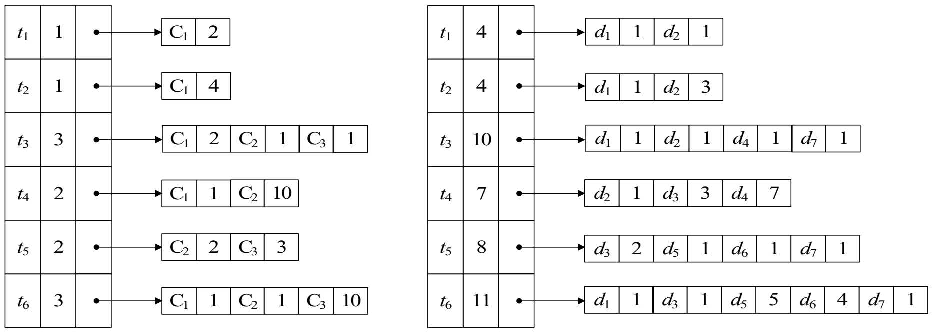


Fig. 1. Centroid and document IIS for typical CBR.

for uncompressed environments and the findings not directly applicable to our framework here.

In Can et al. [2004], we have proposed a method to improve typical-CBR performance by using a skipping, inverted index. In this structure, cluster membership information is also blended into the inverted index, and those posting-list entries that are not from best clusters are *skipped* during query processing.

In this article, we propose a new IIS file that accommodates centroid- and document posting lists in a fully combined manner, and an incremental-CBR strategy to access this IIS file efficiently. This article differs from our earlier work [Can et al. 2004] in the following ways.

—The cluster-skipping IIS file is enhanced to allow both centroid-query matching and document-query matching at once.
—A new incremental query processing strategy is introduced to be used with this index file.
—Efficiency results are provided for both in-memory- and disk-access times during query processing.
—The method is essentially proposed for and adapted to environments with compression.

2.3 Compression of IIS

There are several works regarding the compression of inverted indexes, and in this section we briefly summarize them based on the discussion in Witten et al. [1994]. The key point for compressing posting lists is storing the document ids in list elements as a sequence of *d-gaps*. For instance, assume that the posting list for a term t includes the following documents: 3, 7, 11, 14, 21, 24; using d-gaps these can be stored as 3, 4, 4, 3, 7, 3. In this representation, the first document id is stored as-is, whereas all others are represented with a d-gap (id difference) from the previous document id in the list. The expectation is that d-gaps are much smaller than the actual ids. Among many possibilities, variable-length encoding schemes are usually preferred to encode d-gaps and term frequencies, as they allow representing smaller integers in less space. There are several bitwise encoding schemes. In this study we will focus on the Elias-γ- and Golomb codes, following the approach implemented in Moffat and Zobel [1996] and Witten et al. [1994]. More recently, Anh and Moffat [2005a] proposed a more efficient compression scheme which could also be applicable in our framework.

In the literature, a particular choice for encoding typical posting-list elements (i.e., $<d, f_{d,t}>$ pairs) is using the Golomb- and *Elias-γ* schemes for d-gaps and term frequency values, respectively [Witten et al. 1994]. Elias-γ code is a non-parameterized technique that allows easy encoding and decoding. Golomb code is a parameterized technique which, for some parameter b, encodes a nonzero integer x in two parts. For inverted index compression, the parameter b can be determined by using a global Bernoulli process that models the probabilistic distribution of document id occurrences in posting lists. Golomb code can be further specialized by using a local Bernoulli model for each posting list. In this case, the d-gaps for frequent terms (with longer posting lists) are coded with small values of b, whereas d-gaps for less frequent terms are coded with larger ones. During encoding and decoding, the b value is determined for a particular posting list I_t by the formula

$$b = 0.69 \times \frac{N}{f_t}. \tag{1}$$

In this equation, N is the number of documents, and f_t is the frequency of term t in the collection (i.e., the length of the posting list I_t). In Witten et al. [1994, pp. 94–95], it is reported that "for most applications . . . the local Bernoulli model, coded using the Golomb code, is the method of choice." In Section 4, we discuss in detail how the Golomb code with a local Bernoulli model can be adapted to cluster-skipping IIS.

3. INCREMENTAL CBR WITH CLUSTER-SKIPPING INVERTED INDEX STRUCTURE

3.1 Cluster-Skipping Inverted Index Structure with Embedded Centroids

A cluster-skipping inverted index structure (CS-IIS) differs from a typical IIS, since in posting lists it adjacently stores the documents of each cluster in a group. It contains a skip element preceding each such group to store the id of that cluster to which the following document group belongs, and a pointer to the address where the next skip-element can be accessed in the posting list. It is shown that cluster skipping in query processing improves the query processing time [Can et al. 2004]. Furthermore, since cluster membership information is embedded into the IIS, it needs no separate cluster membership test, as required in typical-CBR methods.

$$D = \begin{array}{c} \\ \\ \\ \\ \\ \\ \\ \end{array}\begin{array}{cccccc|c} t_1 & t_2 & t_3 & t_4 & t_5 & t_6 & \\ 1 & 1 & 1 & 0 & 0 & 1 & d_1 \\ 1 & 3 & 1 & 1 & 0 & 0 & d_2 \\ 0 & 0 & 0 & 3 & 2 & 1 & d_3 \\ 0 & 0 & 1 & 7 & 0 & 0 & d_4 \\ 0 & 0 & 0 & 0 & 1 & 5 & d_5 \\ 0 & 0 & 0 & 0 & 1 & 4 & d_6 \\ 0 & 0 & 1 & 0 & 1 & 1 & d_7 \end{array}$$

$C_1 = \{d_1, d_2\}$
$C_2 = \{d_3, d_4\}$
$C_3 = \{d_5, d_6, d_7\}$

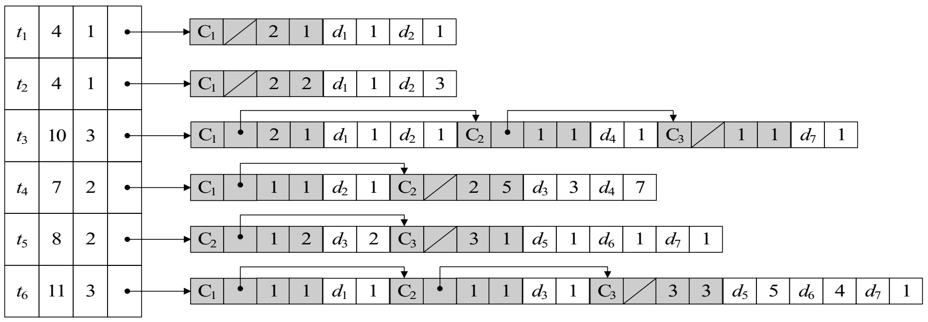

t1 4 1 → C1 2 1 d1 1 d2 1
t2 4 1 → C1 2 2 d1 1 d2 3
t3 10 3 → C1 2 1 d1 1 d2 1 C2 1 1 d4 1 C3 1 1 d7 1
t4 7 2 → C1 1 1 d2 1 C2 2 5 d3 3 d4 7
t5 8 2 → C2 1 2 d3 2 C3 3 1 d5 1 d6 1 d7 1
t6 11 3 → C1 1 1 d1 1 C2 1 1 d3 1 C3 3 3 d5 5 d6 4 d7 1

Fig. 2. Cluster-skipping inverted index structure (CS-IIS) example (embedded skip- and centroid elements are shown as shaded).

In this article, we introduce a new cluster-skipping IIS which contains an additional centroid element for each cluster in a given posting list (see Figure 2). In this illustration, we use the same D matrix shown in Figure 1 and repeat it here for convenience. Note that in both Figures 1 and 2, the same D matrix and clusters of Can et al. [2004] are used to emphasize similarities and differences between the two skip approaches. The new centroid-element stores: (i) the number of documents (i.e., subposting-list length, explained later), and (ii) average within-cluster term frequency for the term in the corresponding cluster. These fields are used during cluster-query similarity computation, and in fact represent those centroids used for the selection of the best-matching clusters. Therefore, in our approach the centroid information is stored with or embedded into document posting lists. In the figure each posting-list header contains the associated term, the number of posting-list elements (pairs) associated with that term, number of clusters containing the term, and the posting-list pointer (disk address). The posting-list elements are of three types: the *cluster id*, which is the position of the next cluster, the *number of documents in the subposting list*, that is, the average within-cluster term frequency, the and *document id*, namely the term frequency. Note that while the latter is a typical posting-list element, the first two are called the *skip element* and *centroid element*, respectively. In a posting list, the skip- and centroid element, along with succeeding typical elements (until the next skip-element), are collectively called a *subposting list*.

In Figure 2, the posting list for t_6 includes documents from three clusters. For the first two clusters, the centroid elements simply store <1, 1> (since the number of documents in cluster C_1 (C_2) is 1), as well as the average within-cluster term frequency. For the last cluster in this posting list, the centroid element is <3, 3>, since there are three documents in cluster (d_5, d_6, d_7) and

```
Input: Query Q, Cluster-skipping IIS (CS-IIS), document lengths, cluster lengths, $n_s$ (no. of best
clusters to be selected), $d_s$ (no. of best documents to be selected)
In-memory data structures: Document accumulator DAcc, cluster accumulator CAcc, best
clusters BestClus

1. Sort the terms of Q in descending (i.e., nonincreasing) order of term weight $w_{q\,t}$
2. For each term t in Q
    a) Retrieve $I_t$, the posting list of term t from CS-IIS.
    // First pass over the posting list: selecting the best-matching clusters
    b) For each subposting list $IS_t$ in $I_t$
        i) Access the skip- and centroid-element <Cid, address> and <sub-posting list length,
           average within-cluster term frequency> from $IS_t$
        ii) Compute $w_{Cid,\,t}$ using subposting list length and average within-cluster term
           frequency
        iii) $CAcc\,[Cid] \leftarrow Cacc[Cid] + w_{q,t} * w_{Cid\,t}$
        iv) Access the next skip-element pointed to by the address
    c) For each nonzero value in CAcc, normalize the computed value (i.e., divide by cluster
       length)
    d) Select $n_s$ best clusters into BestClus that have the highest CAcc scores (by using a
       min heap)
    // Second pass over the posting list: selecting the best-matching documents
    e) For each subposting list $IS_t$ in $I_t$
        Access the skip-element <Cid, address> from $IS_t$
        If Cid is in BestClus
            For each document in $IS_t$
                Access the typical element <document id Did, term frequency $tf_t$>
                Compute $w_{Did\,t}$ using $tf_t$
                $DAcc\,[Did] \leftarrow DAcc\,[Did] + w_{q,t} * w_{Did\,t}$
        Else
            Access the next skip-element pointed to by the address
3. For each nonzero value in DAcc, normalize the computed value (i.e., divide by document
length)
4. Select $d_s$ best documents that have the highest DAcc scores (by using a min heap)
```

Fig. 3. Evaluation of a ranking-query by incremental CBR with embedded centroids, using cluster-skipping IIS.

the average within-cluster term frequency (as an integer) in the cluster is $(5 + 4 + 1)/3 = 3$.

An immediate benefit of this new inverted index structure is that there is no need for a separate centroid index, and subsequently no need for an additional direct disk-access time per query term for fetching the centroid IIS posting list (assuming that the latter would reside on disk). By embedding cluster information into the posting lists, any term in a cluster (or all of the terms) can be chosen as a centroid term and during the query processing its weight can be computed by using the methods described in Section 2.2.1. For simplicity, assume all terms that appear in a cluster are used in the cluster centroids. In this case, the within-cluster term frequency of the term is required to compute the *tf* component of the term weighting schemes (e.g., CW2 and CW3 of Table I). This value is approximately computed as the product of the values stored in the centroid element in a subposting list (i.e., *subposting-list length* $\times$ *average within-cluster term frequency*), as shown in step 2(b)-(ii) of Figure 3.

Note that instead of storing the actual within-cluster term frequency in the centroid element, we prefer to store the average frequency value and obtain the actual value by a multiplication. This is for the benefit of the compression process (discussed in the next section), as smaller integers occupy less space during compression. We expect that using an approximate value, instead of the actual within-cluster term frequency in a cluster, does not affect overall system effectiveness, which is justified by the experimental results. For the *idf* component of weighting schemes, the number of clusters including a term is required. Notice that this information is captured in the IIS header (see Figure 2).

Note that we assume the cluster lengths (i.e., centroid normalization factors used in matching [Salton and Buckley 1988]) to be precomputed and stored just like document lengths, for whichever term weighting scheme is used. During query processing, centroid-term weights are normalized by using the precomputed cluster lengths.

3.2 Incremental Cluster-Based Retrieval

In incremental CBR, we determine the best clusters by only accessing the cluster-skipping IIS. The basic heuristic is that instead of determining the final best clusters before ranking the documents in these clusters (as in the case of typical CBR), we carry out both processes in incremental fashion. In this new strategy, the query terms are processed in decreasing order according to their weight. For a given query, the posting list for the most important query term is brought to memory. In the first pass over its posting list, the *best-clusters-so-far* are determined using an appropriate centroid-term weighting scheme (see Section 2.2.1) and similarity measure. Notice that the information required for these schemes is available in the skip- and centroid elements (as mentioned in the previous section), so during the first pass it is sufficient to access *only* those elements of each subposting list. In the second pass, only those documents whose clusters fall into the best-clusters-so-far are considered, while the system skips those documents from the nonbest-matching clusters, as before. The same is repeated for the next term in order (see Figure 3). Remarkably, during query processing, only necessary elements of the CS-IIS are accessed in each pass. This is especially important for reducing the number of decoding operations in a compressed environment.

For instance, assume a query containing the terms $\{t_4, t_6\}$ and that the number of best clusters (n_s) and number of best documents (d_s) to be selected are 2. Further, assume that t_4 has a higher term weight than t_6 for this query (see Figure 4). Then, first the posting list of t_4 is fetched. In the first pass, the query processor reaches only the skip- and centroid elements in the posting list and updates the cluster accumulator entries for C_1 and C_2. Let us assume that their similarity scores are (partially) computed as 0.65 and 0.75, respectively. Then, since the number of best clusters to be selected is 2, these two clusters will be in the best-clusters-so-far, and in the second pass the document accumulator entries for the documents in these clusters (i.e., documents d_2, d_3, d_4), will be updated (say, as 0.1, 0.3, 0.7, respectively). Next, the posting list of t_6 is fetched. Let us assume that this updates the cluster accumulator entries for clusters C_1,

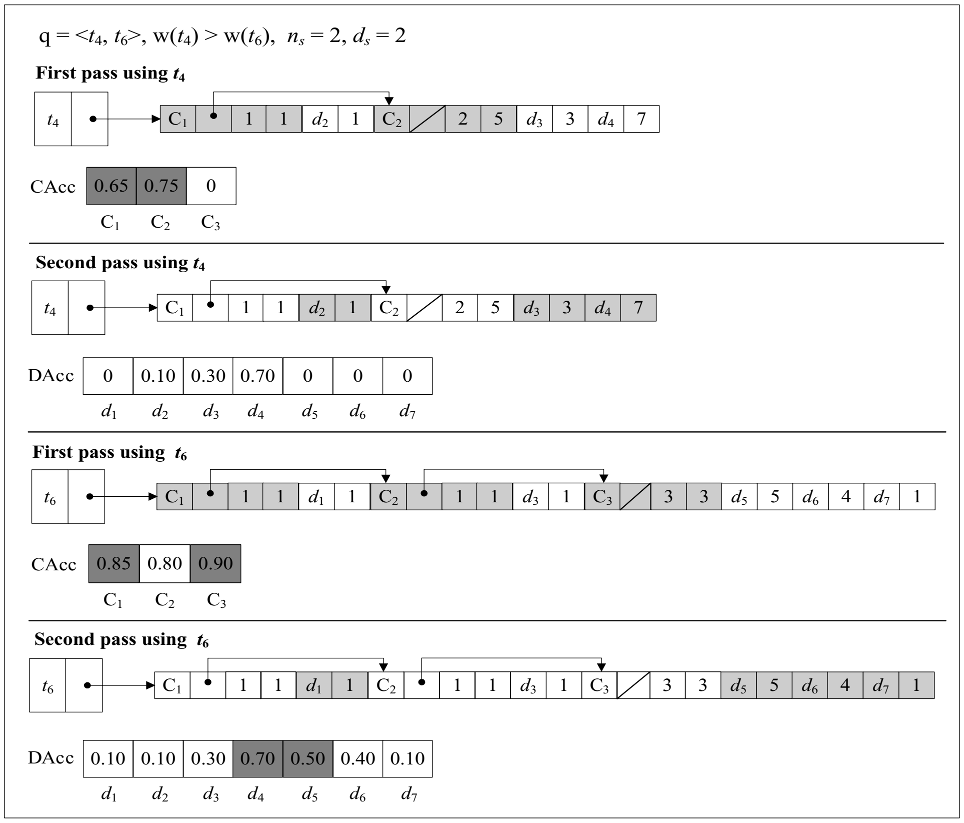

q = <t4, t6>, w(t4) > w(t6), ns = 2, ds = 2
First pass using t4
t4
C1 1 1 d2 1 C2 2 5 d3 3 d4 7
CAcc 0.65 0.75 0
C1 C2 C3
Second pass using t4
DAcc 0 0.10 0.30 0.70 0 0 0
d1 d2 d3 d4 d5 d6 d7
First pass using t6
t6
C1 1 1 d1 1 C2 1 1 d3 1 C3 3 3 d5 5 d6 4 d7 1
CAcc 0.85 0.80 0.90
Second pass using t6
DAcc 0.10 0.10 0.30 0.70 0.50 0.40 0.10

Fig. 4. Example query processing using an incremental-CBR strategy (accessed and decompressed list elements are shown with light gray, best documents and clusters are shown with dark gray).

C_2, and C_3 with the additional values 0.20, 0.05, and 0.90, respectively. Now, the best-clusters-so-far set includes C_1 and C_3 with scores 0.85 and 0.90, whereas C_2 with score 0.80 is out; thus, the documents from the former two clusters are considered, but the subposting list for C_2 is skipped during the second pass. In other words, the documents d_1, d_5, d_6, and d_7 will be updated (say, as 0.1, 0.5, 0.4, and 0.1, respectively). The highest-ranking two documents, d_4 and d_5, are returned as the query output.

In summary, using the proposed incremental-CBR strategy with the CS-IIS file has two major advantages: First, both embedding the cluster information into the IIS and using the incremental query evaluation method eliminates the need for a separate centroid IIS, and hence the disk-access time to retrieve its posting lists. This means that in a memory-scarce environment where the index files are kept on disk, incremental CBR achieves half the number of direct disk accesses as required by typical CBR, and the same number as required by FS. Second, cluster skipping and thereby decoding only relevant portions of the CS-IIS during both stages of query processing saves significant decompression overhead. This means improved in-memory query processing performance with respect to typical CBR and FS. In the next section, we discuss how we handle the only overhead of CS-IIS, that is, the storage consumption due to newly added skip- and centroid elements, by adapting the compression techniques in the literature.

4. COMPRESSION AND DOCUMENT ID REASSIGNMENT FOR CLUSTER-SKIPPING IIS

4.1 Compressing Cluster-Skipping IIS

As discussed before, cluster-skipping IIS includes three types of elements in posting lists: (i) the skip elements in the form of cluster id (position of the next cluster); (ii) the centroid elements in the form of subposting-list length (average within-cluster term frequency); and (iii) the typical elements of the type called document id (term frequency). For the compression of such a posting list, we consider three types of gaps: *c-gaps* between the cluster ids of two successive subposting lists, *a-gaps* between address fields (i.e., following the approach taken in Moffat and Zobel [1996]), and the typical *d-gaps* for document ids.

Example. Let's consider the posting-list entry for t_3 of Figure 2, in which skip- and centroid elements are shown in bold.

<1, add2> <2, 1> <1, 1> <2, 1> **<2, add3> <1, 1>** <4, 1> **<3, EOL> <1, 1>** <7, 1>

The list to be compressed will be represented as follows.

<1, add2> <2, 1> <1, 1> <1, 1> **<1, add3-add2> <1, 1>** <4, 1> **<1, EOL> <1, 1>** <7, 1>

Note that the underlined fields are represented as gaps. End of list (EOL) is represented by the smallest possible integer that can be compressed, namely, 1.

There are two subtle issues regarding the aforesaid representation. Assume that d-gaps are encoded by using the Golomb code with the local Bernoulli model, which is a common practice in the literature [Witten et al. 1994]. In this case an appropriate way of computing the Golomb parameter b is required, since the original formulation does not consider that documents in our CS-IIS are grouped together according to their clusters (i.e., into subposting lists) and that the document id distribution probability must be revised to reflect this modification as well. As a simple solution, for posting list I_t for term t, we revise the formula as follows, assuming that the documents with term t are uniformly distributed among those clusters that appear in I_t. The value "no. of clusters" is assumed to be stored with the wordlist (header) of the IIS (see Figure 2).

$$b = 0.69 \times \frac{N}{f_t/\text{no. of clusters in } I_t} \tag{2}$$

The second important observation for the aforesaid representation is that, for the cluster-skipping IIS, the first document id in *each* subposting list per cluster (e.g., d_1, d_4, and d_7 in the earlier example) should be encoded as-is, which may significantly diminish the compression ratio. In the next section, we propose a remedy for this problem.

4.2 Document ID Reassignment

Document id reassignment is an emerging research topic that attempts to make the document ids in a posting list as closely aligned as possible, so that the frequency of small d-gaps improves compression rates [Silvestri et al. 2004]. In the cited work, it is also reported that some other techniques (as in Blanford and Blelloch [2002]) can provide better compression rates (i.e., up to 10% smaller) with respect to a cluster-based scheme as described in the following. However, we prefer to use the cluster-based reassignment method, which can be amortized by and computed during the clustering process.

Here, we apply an apparently natural document id reassignment method: Essentially, documents in the same cluster are assigned consecutive ids, and the order among clusters is determined according to their creation order by the clustering algorithm. Similarly, the order of documents in a cluster is determined by their order of entrance into the cluster. Notice that a similar approach using a k-means clustering algorithm is reported in Silvestri et al. [2004], among many other techniques.

For CS-IIS, the expected benefit of document id reassignment is twofold: (i) In each subposting list per cluster, the d-gaps between successive documents' ids are reduced; and (ii) more importantly, the id of the first document (which must be encoded as-is) in each subposting list can be reassigned a smaller value. Indeed, with a little main-memory consumption, it is possible to amplify the benefit mentioned in item (ii) significantly. In each cluster, documents are assigned a *real id* which is determined as described before, and a *virtual id* which starts from 1 and increments by 1, to be used just for compression purposes. During compression, virtual ids are compressed such that each subposting list will start with a considerably smaller id than the original. During query processing, an array is kept in main memory to store the prefix sum of cluster sizes, the so-called *size-sum array*. Whenever a document id field is decoded, the decoded virtual id is added to the prefix sum value stored for this document's cluster (which is already known, since decoding starts from the skip element per subposting list) in the size-sum array to obtain the real id. Subsequently, the corresponding correct document accumulator is updated for this real id.

Example. Assume that cluster C_1 includes two documents and cluster C_2 includes three. The documents from C_1 and C_2 will be assigned to real ids 1, 2, and 3, 4, 5, respectively. The virtual ids are also 1 and 2 for C_1, but 1, 2, and 3 for C_2. The size-sum array will store 0 for C_1, $0 + \text{size of}\,(C_1) = 0+2 = 2$ for C_2. During query processing, if a document id in C_2's subposting list is decoded as 2, it will be added to the size-sum array value for C_2, which is also 2, to obtain the real id as 4.

Note that the number of clusters would be smaller than the number of documents in orders of magnitudes, so that storing the size-sum array in memory is not a major problem. Furthermore, the array can be kept in the shared memory and accessed by several query processing threads at the same time, that is, it is *query invariant*. Finally, if the Golomb code is employed for encoding d-gaps, the b parameter should be further revised. In particular, we refine it as in Eq. (3), since the virtual documents' ids in each subposting list can range

Table II. Characteristics of the Datasets

Dataset	Size on Disk	No. of Documents (N)	No. of Terms (n)	No. of Clusters	No. of <doc id, term Freq.> Pairs	Avg. no. of Docs/Clusters
FT	564MB (text)	210,158	229,748	1,640	29,545,234	128
AQUAINT	3GB (text)	1,033,461	776,820	5,163	170,004,786	200
WEBBASE	140GB (HTML)	4,293,638	4,290,816	13,742	790,291,775	312

from 1 to "average cluster size" on the average.

$$b = 0.69 \times \frac{\text{average cluster size}}{f_t / \text{no. of clusters in } I_t} \tag{3}$$

As another alternative, we can define a dedicated b value to compress each subposting list separately; see Eq. (4). Note that the cluster size C_i can be easily computed from the size-sum array as the difference in array entries for $i+1$ and i, without requiring an extra data structure. The number of occurrences of t (f_t) in C_i is captured in the centroid element of IS_i (i.e., subposting-list length) and will be decoded immediately before the decoding of d-gaps starts. In Section 6, we evaluate and compare the storage figures for various compression schemes and parameters.

$$b = 0.69 \times \frac{\text{size}(C_i)}{f_t \text{ in } C_i} \tag{4}$$

5. EXPERIMENTAL ENVIRONMENT

5.1 Datasets and Clustering Structure

In the experiments, three datasets are used. The *Financial Times* collection (1991–1994) of TREC [TREC 2006] Disk 4, referred to as the FT dataset, and the *AQUAINT* corpus of *English News* text, referred to as the AQUAINT dataset, have been used in previous TREC conferences and include the actual data, query topics, and relevance judgments. As a third dataset, we obtained the crawl data from the Stanford WebBase Project Repository [Stanford University 2007]. This latter dataset, referred to as WEBBASE, includes pages collected from U.S. government Web sites during the first quarter of 2007. As there are neither query topics nor relevance judgments for this dataset, it is solely used for evaluating query processing efficiency. During the indexing stage, we eliminate English stop-words and index the remaining words, and no stemming is used. For the WEBBASE dataset, words that appear in only one document are also removed, as the Web pages include a high number of mistyped words. In Table II, we provide statistics for the datasets and the indexing results. Notice that the original WEBBASE dataset spans more than 140GB on disk in HTML. After preprocessing and removing all HTML tags, scripts, white spaces, etc., the pure text on disk (tagged in TREC style) takes 27GB.

The datasets are clustered using the C^{3}M algorithm [Can and Ozkarahan 1990] in partitioning mode, which yields 1,640 and 5,163 and 13,742 clusters for the FT, AQUAINT, and WEBBASE datasets, respectively. An important parameter for CBR is the number of best-matching clusters. In earlier works it

Table III. Query Sets Summary Information

Dataset & Query Sets	No. of Queries	Avg. no. of relevant Documents	Query Type	Average no. of Terms	Min no. of Terms	Max no. of Terms
FT, *Qset1*	47	31.8	*Qshort*	2.5	1	4
			Qmedium	10.8	4	30
FT, *Qset2*	49	38.1	*Qshort*	2.4	1	3
			Qmedium	8.2	2	19
			Qlong	190.0	13	612
FT, *Qset3*	49	33.4	*Qshort*	2.4	1	3
			Qmedium	7.3	3	19
AQUAINT, *Qset1*	50	131.2	*Qshort*	2.5	1	4
			Qmedium	9.4	4	20
WEBBASE, *Qset1*	50,000	N/A	*Qshort*	2.3	1	9

has been reported that the effectiveness increases up to a certain n_s value and that after this (saturation) point, the retrieval effectiveness remains the same or improves very slowly for increasing n_s values [Can et al. 2004]. This saturation point is found to be around 10–20% in the literature [Can and Ozkarahan 1990; Salton 1975, p. 376]. Therefore, in the retrieval experiments reported in Section 6, we use 10% of the total number of clusters as the number of best clusters to be selected (i.e., n_s is 164 and 516 and 1374 for the corresponding datasets). In this article, we provide results for retrieving top-1000 documents; that is, the number of best documents to be selected, d_s, is 1000.

The clustering of the largest dataset (WEBBASE) takes around 12 hrs. using a rather out-dated implementation of the C^{3}M algorithm. Once the clustering is completed, creating the typical IIS and CS-IIS takes almost equal time, which is around 6 hrs. for this dataset, by again using an unoptimized implementation. Nevertheless, any partitioning-type clustering algorithm could be used in our setup, provided it can provide reasonable effectiveness by accessing a relatively small percentage of all clusters.

5.2 Query Sets and Query Matching

For the FT dataset, we used three different query sets, along with their relevance judgments that were obtained from the TREC Web site [TREC 2006]. The three query sets, referred to as *Qset1, Qset2*, and *Qset3,* include TREC queries 300–350, 351–400, and 401–450, respectively. Note that the relevance judgments for some of the queries in these sets refer to documents from datasets other than the ones used in this article. Such irrelevant judgments were eliminated, and for each query set we produced a relevance judgment file which includes only documents from the FT dataset. A few of the queries do not have any relevant documents, and they were discarded from the query sets. Table III shows the remaining number of queries for each query set of FT. For the AQUAINT dataset, we used the topics and judgments used for the TREC 2005 robust track. Finally, for the WEBBASE dataset, the efficiency task topics of the TREC 2005 terabyte track were employed. Note that this query set was used on top of the TREC GOV2 dataset, which also includes Web data from the "gov" domain. Since the WEBBASE collection also captures the same domain, we presume this query set to be a reasonable choice for efficiency evaluation with WEBBASE.

In the experiments, we used two different types of queries, namely *Qshort* and *Qmedium*, that are obtained from the query sets discussed previously. *Qshort* queries include TREC query titles, and *Qmedium* queries include both titles and descriptions. For one of the FT query sets (FT-*Qset2*), we also formed a third query type called *Qlong*, which is created from the top retrieved document of each *Qmedium* query in this query set. Our query sets cover a wide spectrum from very short Web-style queries (the *Qshort* case) to extremely long ones (the *Qlong* case). Notice that the latter type of queries can capture the case where a user likes to retrieve documents similar to a particular document and where the document itself serves as a query. This provides insight on the behavior of the retrieval system at extreme conditions. Table III provides the query sets' summary information.

In the following experiments, the document-term weights are assigned using the $tf \times idf$ formula. The cosine function is employed for both query-cluster- and query-document matching. Please refer to Section 2.1 for further details.

5.3 Cluster Centroids and Centroid-Term Weighting

For the cluster centroids, we take a simplistic approach and use all cluster member documents' terms as centroid terms. One reason for this choice is that our experiments in an earlier work [Can et al. 2004] with the FT dataset and FT-*Qset2* have shown the effectiveness not to vary significantly for centroid lengths 250, 500, and 1000, whereas using all cluster terms in the centroid yields slightly better performance. Another reason is that by using all cluster terms in centroids, we avoid making an arbitrary decision to determine the centroid length. This choice of centroids also enables our being independent of a particular centroid-term selection method. Nevertheless, it is possible to apply other centroid-term selection schemes in our framework as well.

The experiments employ the three centroid weighting schemes described in Section 2.2.1. Recall that the information stored in the CS-IIS file is adequate to compute all three schemes, as mentioned in Section 3. As for the documents, we assume that during query processing the weights are normalized by using the precomputed cluster lengths.

6. EXPERIMENTAL RESULTS

The experiments are conducted on a Pentium Core2 Duo 3.0 GHz PC with 2GB memory and 64-bit Linux operating system. All IR strategies are implemented using the C programming language and source-codes are available on our Web site.[1] Implementations of the IR strategies are tuned to optimize the query processing phase for which we measure the efficiency in the following experiments. In particular, a min heap is used to select best clusters and best documents from the corresponding accumulators, as recommended in previous work [Witten et al. 1994]. Unless stated otherwise, we assume that the posting list per query term is read into main memory, processed, and then discarded;

[1]http://www.cs.bilkent.edu.tr/~ismaila/incrementalCBR.htm

that is, one term's posting list is not memory resident simultaneously with another. The document- and cluster-lengths are precomputed.

In what follows, we first compare the effectiveness figures of the incremental-CBR strategy with those of FS and typical CBR, to demonstrate that the new strategy does not deteriorate the quality of query results. Next, we focus on the efficiency of the proposed strategy and show incremental CBR to be better than FS in total query processing performance (involving in-memory evaluation and disk accesses), with a reasonable overhead in storage requirements. Finally, we show that incremental CBR is superior to not only a basic implementation of FS, but also to a faster approach that employs the "continue" pruning strategy along with a skip-embedded IIS, as described in Moffat and Zobel [1996].

6.1 Effectiveness Experiments

To evaluate the effectiveness of the proposed strategy: the top-1000 (i.e., $d_s = 1000$) documents are retrieved for each of the query sets. The effectiveness results are presented by using a single mean average precision (MAP) value (i.e., average of the precision values observed when a relevant document is retrieved) [Buckley and Voorhees 2000] for each of the experiments. All MAP scores are computed using the "trec eval" software [TREC 2006] and the resulting files corresponding to the previous table are available at our Web site (see Footnote 1).

In this section, we compare three IR strategies. FS, typical CBR, and incremental CBR with CS-IIS. For both CBR techniques, all terms in clusters serve as centroid terms. We experiment with three centroid-term weighting schemes (CW1, CW2, and CW3) as described in Section 2.2.1.

The effectiveness results obtained for FS experiments are compared to those obtained when using the publicly available search engine Zettair [2007] so as to verify the validity of our findings and robustness of our implementation. The indexing and querying stages with Zettair are achieved under almost the same conditions as in our own implementations. During indexing, no stemming is used. In query processing, the same stop-word list as we use in our system is provided to Zettair and the cosine similarity measure is chosen. For each dataset, *Qshort*- and *Qmedium* query types are evaluated by retrieving top-1000 results. We found that in almost all experiments our MAP values are slightly better than those of Zettair, which validates our implementation. The details of the experimental procedure and evaluation files are also available at our Web site (refer to Footnote 1).

The first observation that can be deduced by a quick glance over Table IV is that for each query set and type, all MAP values are very close to each other (the best ones are shown in bold). Thus, it is hard to claim that one single strategy totally outperforms the others. Still, the results demonstrate that CBR is a worthwhile alternative to FS for accessing large document collections.

From the aforementioned results it is clear that the proposed strategy has no adverse effect on CBR effectiveness and, in particular cases, can even improve it. In particular, Table IV reveals the incremental-CBR strategy to be better than typical CBR for the majority of the cases, although the absolute MAP

Table IV. MAP Values for Retrieval Strategies

			Typical-CBR			Incremental-CBR		
Dataset & Query Sets	Query Type	FS	CW1	CW2	CW3	CW1	CW2	CW3
FT, *Qset 1*	*Qshort*	0.161	0.162	**0.168**	0.154	0.163	0.167	0.166
	Qmedium	0.152	**0.173**	0.148	0.143	0.158	0.153	0.155
FT, *Qset 2*	*Qshort*	0.107	0.126	0.109	0.102	**0.131**	0.110	0.110
	Qmedium	0.122	0.134	0.121	0.113	**0.137**	0.120	0.120
	Qlong	**0.124**	0.113	0.114	0.109	0.119	0.120	0.119
FT, *Qset 3*	*Qshort*	**0.154**	0.142	0.144	0.131	0.134	0.150	0.147
	Qmedium	**0.170**	0.150	0.166	0.123	0.159	0.161	0.142
AQUAINT, *Qset 1*	*Qshort*	**0.091**	0.046	0.081	0.071	0.047	0.081	0.077
	Qmedium	**0.100**	0.048	0.089	**0.074**	0.057	0.090	0.081

Here, $n_s = 164$ for FT, $n_s = 516$ for AQUAINT, $d_s = 1000$.

improvement is rather marginal. For *Qshort* and *Qmedium* query types of *Qset2* on the FT dataset, the incremental CBR strategy yields the best effectiveness figures, outperforming both FS and typical CBR. Another interesting observation is that for the CBR strategies, CW1 and CW2 are the most promising centroid-term weighting schemes.

We conduct a series of matched pair t-tests to determine whether incremental- and typical-CBR strategies with CW1, CW2, and CW3 are as effective as FS. The null hypotheses in this case would be that the effectiveness of each of these methods is as good as FS, and the alternative is that they are not. For this purpose, we examine the performance differences of these two approaches as provided in Table IV. Note that we are performing one-sided t-tests so we would divide the two-sided p-value by 2. Since we are also performing six hypothesis tests, we perform a Bonferonni correction by multiplying each p-value by 6. Thus, combining the two adjustments, we end up multiplying each two-sided p-value by 3; hence, a significant result would be a p-value of less than $0.05/3 = 0.016$. Each difference is the average of the CBR method subtracted from the full search (FS) for each query type. Since the average differences are negative, on average, FS outperforms each cluster method in terms of precision. However, the only significant difference (based on p-values) is in CW3 between typical CBR and FS ($p < 0.01$). In this case, FS significantly outperforms typical CBR. However, in the other tests there is a lack of evidence that FS significantly outperforms CBR. Since CBR with both CW1 and CW2 outperforms both FS for some query types, CBR has the potential of being as effective as FS.

Finally, it should be emphasized that the incremental-CBR strategy with CS-IIS is not at all intended to improve the effectiveness of CBR, but aims to improve efficiency without deteriorating the effectiveness of typical CBR while providing compatible effectiveness with FS. Recall the recent proposals to improve CBR effectiveness [Liu and Croft 2004] that can obviously be applied in our framework as well.

6.2 Efficiency Experiments

In the following experiments typical CBR is omitted, since by definition (see Section 2.2.2) it already involves FS in the best-document selection stage.

Table V. IIS File Sizes (in MBs) for FS and Incremental CBR

	FS					Incremental-CBR (CW2-3)				
	Uncomp.	Golomb (LB)		Elias-γ		Uncomp. CS-	Golomb (LB)		Elias-γ	
Dataset	IIS file size	OrgID	ReID	OrgID	ReID	IIS file size	OrgID	ReID	OrgID	ReID
FT	225	34	33	44	**43**	343	84	45	105	**50 (14% > FS)**
AQUAINT	1,360	211	209	236	**216**	1,900	520	254	602	**250 (16% > FS)**
WEBBASE	6,322	1,076	1,079	767	**770**	7,362	2,315	968	1,745	**844 (10% > FS)**

Here, LB: local Bernoulli model; OrgID: original document ids; ReID: reassigned document ids.

Subsequently, FS serves as a lower bound for the cost of typical CBR. For the efficiency experiments, we report the results obtained by using all three datasets and corresponding query sets, as shown in Table II. However, to save space, for the FT dataset we only use *Qset2*, for which the effectiveness of incremental CBR also peaks.

6.2.1 *Storage Efficiency.* In Table V, we provide compressed file sizes for the evaluated IR strategies. In particular, the term frequency values in typical IIS and in both fields of the skip- and centroid elements in CS-IIS are encoded with Elias-γ code. The values' d-gaps are encoded by using Elias-γ and Golomb codes in separate experiments. This is due to the observation that one of the schemes, namely the Golomb code, appears unaffected by the document id reassignment methods for typical IIS.

Table V reveals that for FT and AQUAINT datasets, when the original documents' ids in the collections are used, the best compression rates for typical index files are achieved by using the Golomb code with LB (using Eq. (1)). For the WEBBASE dataset, however, Elias-γ performs better (i.e., 767- versus 1,076MB). We attribute this fact to the observation that in the latter dataset, which is yielded by a crawling session, the original document ids are sorted in URL order that exhibits strong locality [Silvestri 2007]. On the other hand, the Golomb code is rather insensitive to such locality and performs best on random distributions [Blandford and Blelloch 2002]. This phenomenon is strongly emphasized by the experiments with reassigned document ids and further discussed next. Nevertheless, for the WEBBASE dataset, the typical IIS size drops from 6,322MB to 1,076MB (17%) and 767MB (13%) with Elias-γ and Golomb (with the LB model using Eq. (2)) schemes, respectively. The compressed IIS sizes also correspond to only 4% and 3% of the uncompressed text document collection (27GB) for the respective cases. This conforms to the results reported in other works in the literature [Witten et al. 1994]. On the other hand, it can be seen that the compression gains on CS-IIS by using original document ids are not as good, and for the WEBBASE dataset the compressed file sizes are 31% and 24% of the uncompressed index using the two compression schemes. However, at this point the potential of document id reassignment, which is naturally applicable for CS-IIS, has not yet been exploited.

Next, we applied the document id reassignment method mentioned in Section 4.2, such that the documents in each cluster have consecutive ids. For this experiment, we first discuss the results when the Golomb code is used to encode d-gaps. Note that the b parameter for LB is set as in Eq. (1) for typical IIS, whereas the enhanced formula derived in Section 4.2 is (Eq. (4)) employed for CS-IIS, so as to reflect the distribution of subposting lists as accurately as possible. Remarkably, the Golomb code with LB provides almost no improvement for the typical IIS, whereas CS-IIS highly benefits from the reassignment. For instance, the size of CS-IIS file for the WEBBASE dataset drops from 2,315- to 968MB, a reduction of more than 50%. This is even less than the compressed size of typical IIS (1,079MB) for the corresponding case. As mentioned before, the insensitivity of typical IIS for reassigned ids is caused by characteristics of the Golomb code which cannot exploit the locality (i.e., Golomb code should still use the same b parameter for LB after reassignment). In particular, for FT and AQUAINT datasets the reductions in compressed index sizes are at most 3%, hardly an improvement. For WEBBASE there is even a slight increase (0.3%) in index size. On the other hand, after reassignment the CS-IIS allows to use an enhanced b parameter (Eq. (4)) and benefits from the reassignment procedure, even when Golomb code is used.

For the sake of fairness, we repeated the experiments with reassigned ids and by encoding d-gaps with the Elias-γ method. In this case, as Table V demonstrates, the typical IIS also obtains some gains from document id reassignment, but these gains are still not very impressive in comparison to CS-IIS. Noticeably, the storage space used for the compressed index files of FT and AQUAINT drops by 2% and 9%, respectively. For WEBBASE there is no improvement in index size, but again a slight increase is observed. This is due to the fact that the original URL-ordering ids for this dataset provides quite strong locality, and the reassignment based on clustering does not further improve the compression rate (a result also shown in Silvestri [2007]). To validate this claim, we assigned random ids to documents in the WEBBASE dataset and repeated the compression experiments. In this case, the compressed index sizes are 1,132- and 1,473MB for the Golomb and Elias-γ methods, respectively. These results support our claims in that: (i) If the original document ids are not sorted in URL order, the Golomb code with LB provides better compression rates (as in the cases of FT and AQUAINT) with respect to Elias-γ; (ii) the Golomb code is rather insensitive to any locality in that the file sizes for random and URL-sorted experiments are very close (1,132- and 1,076MB, respectively), whereas Elias-γ is just the reverse (i.e., the index size drops from 1,473- to 776MB); and (iii) sorting by URL order provides a very good d-gap distribution, as shown by the results of Elias-γ, and the typical IIS size cannot be reduced by further reassignment. In contrast, CS-IIS still significantly benefits from id reassignment (i.e., yielding reductions of more than 50% in size). For instance, by using the Elias-γ encoding method, the CS-IIS file for WEBBASE only takes 844MB on disk, which is only 10% larger than the typical IIS for the corresponding case (770MB). This is a striking result for the space utilization of CS-IIS that is obtained by using a cluster-based document id reassignment technique, and is a natural advantage of our framework.

Recall that the document reassignment method for CS-IIS employs virtual ids instead of real ids in the subposting lists, so as to encode the first document of each subposting list more efficiently (see Section 4.2). We devised a separate experiment to evaluate the performance of this heuristic. For the WEBBASE dataset, we simply reassigned document ids. In this case, the first document

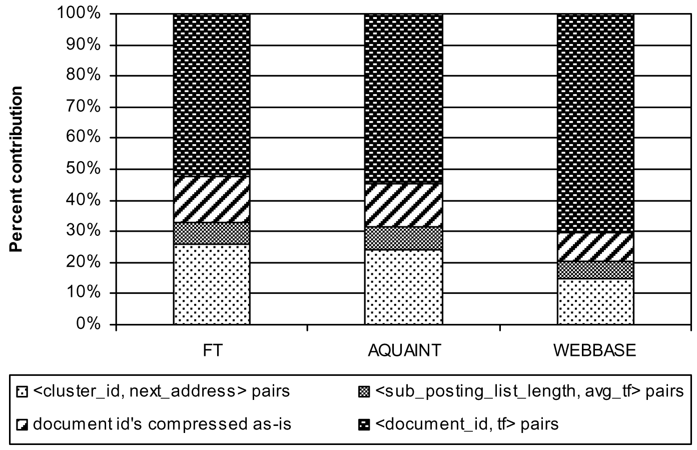


Fig. 5. Contribution of CS-IIS posting-list elements to compressed file sizes for the three datasets.

id of each posting list, which should be compressed as-is, takes 330MB and 232MB of the resulting CS-IIS file for the Elias-γ and Golomb code with LB (using Eq. (1)), respectively. Next, we applied the optimization of Section 4.2 (i.e., virtual ids are assigned within each cluster to reduce the actual value of the first document ids in subposting lists). In this case, only 100 MB and 76 MB of CS-IIS is devoted to first ids, again for the Elias-γ and Golomb code with LB, respectively. For the latter scheme, the b parameter for Golomb is now set as in Eq. (4), which is a unique opportunity allowed by CS-IIS. Notice that for both compression schemes, our optimization reduces the space used for first ids to almost one-third of the original space. Moreover, our formulation for the b parameter allows the Golomb code to provide a much better compression ratio with respect to Elias-γ, leading to a further 24% reduction in size. This experiment shows that efficient compression of the first document id in each subposting list of CS-IIS is important for the overall compression efficiency, and the heuristic outlined in this article provides significant gains. Therefore, in all reassigned id experiments for CS-IIS (as reported in Table V), the first document ids are always encoded with Golomb code, regardless of the schemes the remaining d-gaps are compressed. Note that in this heuristic, the size-sum array (of size number of clusters) takes only a few KBs of in-memory space, even for WEBBASE, which is a negligible cost.

In summary, by using a cluster-based id reassignment approach, both Golomb coding with the LB model and the Elias-γ scheme prove quite successful for compressing CS-IIS. Remarkably, by using the Elias-γ scheme, the additional cost of storing CS-IIS, with respect to typical IIS, is at most 16% (see the last column of Table V). In the remaining experiments, we use the compressed typical IIS and CS-IIS files obtained by the id reassignment and the Elias-γ encoding for d-gaps (i.e., those shown in bold in Table V).

In Figure 5, we provide the percentage of bits used to encode each field in the compressed CS-IIS file (for the file sizes in the last column of Table V). Considering the figure, we realize that for WEBBASE, 70% of the file is used to store

Table VI. Efficiency Comparison of FS and Incremental CBR (times in msec.)

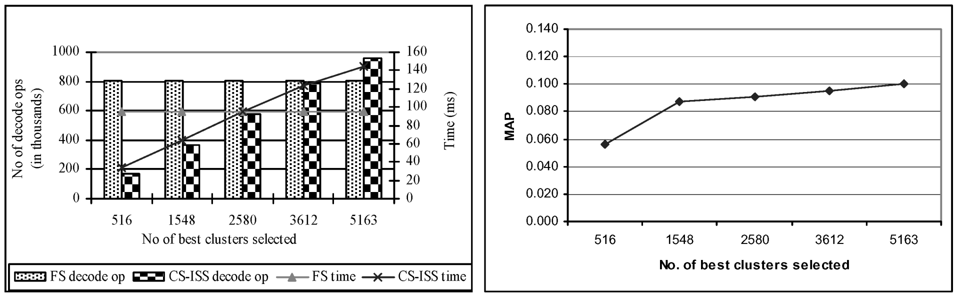
Data and Query Set	Query Type	Execution Time & No. of Decode Op. (averages)	FS	Incremental-CBR		Imp. over FS (%)	
				CW1	CW2	CW1	CW2
FT, *Qset2*	*Qshort*	Exe. time	5	3	4	40	20
		Decode op.	19,524	**8,212**	11,614	58	41
	Qmedium	Exe. time	16	7	9	56	44
		Decode op.	98,832	**36,701**	51,772	63	48
	Qlong	Exe. time	389	144	222	63	43
		Decode op.	3,627,468	**1,091,212**	2,079,408	70	43
AQUAINT, *Qset1*	*Qshort*	Exe. time	27	15	19	44	30
		Decode op.	162,824	**37,860**	73,249	77	55
	Qmedium	Exe. time	95	34	48	64	49
		Decode op.	802,740	**172,415**	313,291	79	61
WEBBASE, *Qset1*	*Qshort*	Exe. time	66	36	57	45	14
		Decode op.	432,238	**87,289**	318,431	80	26

actual <document id, tf> pairs, whereas 15% is used for the skip elements (i.e., in the form of <cluster id, next address>) and 5% for centroid elements (i.e., in the form of <sub- posting list length, avg. tf>). Since each subposting list encodes its first document as-is, a considerable fraction of the file (around 10%) is used for this purpose. Notice that while our extra posting-list elements cause 30% of the overall cost in CS-ISS, they also allow the document id reassignment to be more efficient, and thus the overall size remains within an acceptable margin of typical IIS. Figure 5 also shows that the percentage of additional storage in CS-ISS reduces as the dataset gets larger, that is, 50%, 45%, and 30% for FT, AQUAINT, and WEBBASE, respectively. Remarkably, these percentages are not necessarily reflected to CS-IIS file size as increments, as discussed before.

Finally, note that the compression process takes the same time for corresponding cases by using our own implementations, ranging from a few mins. (for FT) to an hr. (for WEBBASE).

6.2.2 *Query Processing Time Efficiency.* In Table VI, we report average CPU (in-memory) processing times per query, as well as the average number of decode operations (i.e., total number of Elias-γ- and Golomb decode operations). The experimental results are provided for CW1 and CW2; the CW3 case is omitted since its efficiency figures are similar to those of CW1.

The results reveal that incremental CBR decompresses a significantly smaller number of elements compared to FS. This is because the former decompresses only relevant portions of a posting list, whereas FS, of course, must decode the entire posting list for a query term (in Section 6.3 we also discuss a skipping-based pruning technique for FS, as discussed in Moffat and Zobel [1996]). For CW1, the savings of incremental CBR in terms of number of decode operations is more emphasized, ranging from 58% to 80% of the decode operations by FS. For CW2, incremental CBR decodes more elements, but the number of decoded elements is still almost half that of the FS case. These savings are reflected to time figures rather conservatively, especially for shorter queries. Time savings improve as queries become longer (e.g., for AQUAINT



(a) (b)

Fig. 6. (a) Effects of the selected best cluster number on processing time and decode operation number; (b) effectiveness (for *Qmedium* using CW1 on AQUAINT dataset).

the savings are 44% (30%) and 64% (49%) for *Qshort* and *Qmedium* using CW1 (CW2), respectively). If we assume that posting lists are kept in the main memory (due to OS caching and large memories), then these savings become final execution-time improvements.

Note that savings in time are not directly proportional to saving in the number of decode operations because the incremental-CBR strategy with CS-IIS has also some overheads, such as jumping to the next bit position to be decompressed and selecting the best clusters from the cluster accumulators for each query term.

In Figure 6(a) we plot the number of best clusters selected versus average number of decode operations (shown on the left y-axis) and average CPU query processing time (shown on right y-axis) for FS and incremental CBR, for *Qmedium* using the CW1 centroid weighting scheme and the AQUAINT dataset. At the extreme, all clusters are selected and incremental CBR degenerates into FS. The number of decode operations realized by incremental CBR as well as the execution time are lower than those of FS until more than 50% of clusters (i.e., greater than 2,580) are selected. Nevertheless, in practical CBR systems, the number of best clusters to be selected comprises a relatively small percentage of the total number of clusters [Can and Ozkarahan 1990; Salton 1975].

In Figure 6(b), we plot the variation in number of best clusters selected versus effectiveness. Note that after 30% of the clusters are selected as best, the MAP figures change slightly. Thus, for the AQUAINT dataset it is possible to set best clusters as 30% of all clusters (i.e., 1,548). Note that even for this case, both the number of decompression operations and the execution time are still significantly less than those for FS, see Figure 6(a). For sake of uniformity, we keep the best clusters as 10% throughout the experiments.

In Table VII we provide the average size of posting lists fetched from the disk during query processing. Both FS and incremental CBR make only one direct access per query term. As expected, incremental CBR fetches slightly longer posting lists with respect to FS (due to the storage overhead of skip- and centroid elements). Note that the increase in posting sizes remains marginal and does not exceed 20%.

Table VII. Avg. Size of Fetched Posting Lists per Query (all in KBs)

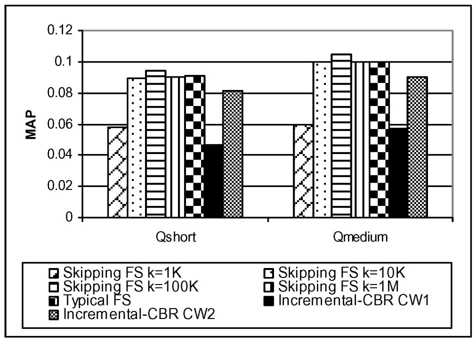
Data and Query Set	Query Type	FS	Incr.-CBR	Overhead over FS (%)
FT, *Qset2*	*Qshort*	10.54	12.56	19
	Qmedium	51.09	60.78	19
	Qlong	1670.77	1962.12	17
AQUAINT, *Qset1*	*Qshort*	73.48	83.55	14
	Qmedium	345.64	391.26	13
WEBBASE, *Qset1*	*Qshort*	147.96	157.75	7

We expect that the cost of these longer sequential accesses would be compensated by the in-memory improvements in decoding times. For instance, assume a (rather slow) disk with the transfer rate 10MB/sec. In this case, the additional sequential-read time cost of CS-IIS with respect to FS for processing a query in the *Qshort* set of WEBBASE would be only around ≈1 msec. (i.e., (157.75–147.96KB /10MB/sec.). For this latter case, FS takes 66 msec. in CPU, whereas incremental CBR takes 36 and 57 msec. for the CW1 and CW2 cases, respectively (see Table VI). Clearly, even with a slow disk, in-memory time improvements are far greater than the disk read overhead (i.e., 30 msec. (for CW1) and 9 msec. (for CW2) versus 1 msec.). Thus, as long as the number of clusters is significantly less than the number of documents, which is a reasonable assumption, our approach would be feasible. Furthermore, assuming that posting lists are kept in the main memory, which is the case for some Web search engines (e.g., Google), our significant performance gains obtained during in-memory query processing show conclusively the improvement.

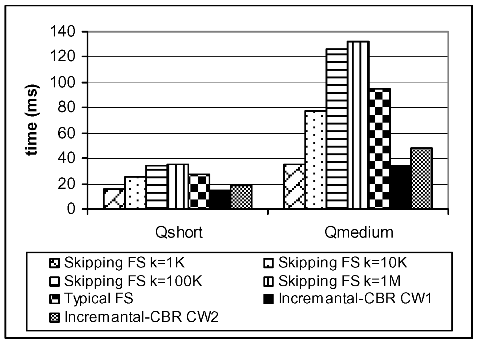
6.3 Experiments with FS Using the Continue Strategy and Skipping IIS

Although we aim to improve CBR as an alternative access method to large document collections, but not as a preprocessing or pruning stage for FS, we also compare our method with a more efficient FS approach using another pruning technique in the literature. In particular, since our approach is inspired from an earlier work that enriches the typical inverted index with skip elements [Moffat and Zobel 1996], it seems to be a natural choice to implement and compare it with our incremental-CBR approach.

In Moffat and Zobel [1996], a posting list has a number of skip elements, each followed by a constant-sized block of typical elements. A skip element has two components: the smallest document id in the following block and the pointer to the next skip-element (and block). This was shown very efficient for conjunctive Boolean queries in a compressed environment. In particular, after the first posting list is processed, a candidate set of document ids are obtained, which are looked for in the other lists. Obviously, when searching to see whether a document is in a particular block, skip elements are very useful: If the document id at-hand is greater than the current skip-element and less than the next, this block is decompressed; otherwise search process jumps to the next skip-element without redundantly decompressing the block. Note that this technique is impossible to be used as-is with the ranking-queries, since there is no set of candidate documents (as in the Boolean case). Therefore, quit- and continue pruning strategies are accompanied with ranking-query evaluation to



(a)



(b)

Fig. 7. (a) Effectiveness of skipping FS vs. no. of accumulators; (b) query processing time of IR strategies.

allow use of the skipping inverted index. Since the effectiveness figures of the continue method are quite close to those of FS without any pruning (referred to as *typical FS* in the following), we prefer to use the continue strategy in this article.

In the continue strategy, the query processor is to allowed to update only a limited number k of accumulators. Until this limit is reached, it decodes the entire posting list for each query term, just like typical FS. After this limit is reached, those nonzero accumulators that have been updated up to this time serve as the candidate document ids in the Boolean case, and are the only accumulators that can be updated. Thus, it is possible to use skip elements and avoid decompressing blocks that do not include any documents with corresponding nonzero accumulators. We refer to this strategy as *skipping* FS.

In Moffat and Zobel [1996], each posting list can have a different number of skip elements, according to the size of posting list and candidate document set [Moffat and Zobel 1996, p. 363]. It is also stated that the continue strategy can achieve comparable (or better) effectiveness figures, even when 1% of the total accumulators are allowed for update. A good choice while constructing the skipping inverted index is assuming that the same k value represents the number of candidate documents for the queries.

In this section we use AQUAINT, the largest dataset with relevance judgments, for the experiments. Since this collection includes approximately 1M documents, k is set to 10,000 (i.e., 1% of the total document number). For the same k value, a skipping IIS is constructed in exactly the same way as described in Moffat and Zobel [1996]. The resulting index file takes 279MB, which is 18% larger than the IIS with no skips (i.e., 236MB, as shown in Table V). In Figure 7(a), the MAP figures using this IIS file and varying the number of accumulators (k) is shown. As expected, the effectiveness figures at $k = 10\text{K}$ are as good as the effectiveness score when all 1M accumulators are available, that is, for typical FS.

In Figure 7(b), CPU execution times for the skipping-FS strategy with a varying number of accumulators are reported (results for typical FS and incremental CBR with CW1 and CW2 are also repeated from Table VI for easy

Table VIII. No. of Decompression Operations for Skipping FS (with varying number of accumulators), Typical FS, and Incremental CBR

Query Type	Skipping FS with Continue Strategy				FS	Incremental-CBR	
	k = 1K	K = 10K	k = 100K	k = 1M		CW1	CW2
Qshort	40,377	106,150	190,556	190,635	162,824	**37,860**	73,249
Qmedium	123,777	408,601	886,858	930,714	802,740	**172,415**	313,291

comparison). Clearly, skipping FS improves the time performance of typical FS, of up to 41% for *Qshort* and 63% for *Qmedium* when $k = 1$K. However, for this case, MAP scores also decrease. For the $k = 10$K case, the improvements of skipping FS are 7% and 19% for *Qshort* and *Qmedium*, respectively. Nevertheless, the performance of incremental CBR (with both CW1 and CW2) remains superior. In Table VIII, we report the number of decompression operations for the corresponding cases. Again, skipping FS improves over typical FS, but cannot match incremental CBR, for the $k = 10$K case.

Finally, note that dynamic pruning techniques such as the one described earlier can also be applied to both typical- and incremental CBR. For instance, during the best-document selection stage of typical CBR, it is possible to embed the skipping-FS approach. Similarly, the skipping strategy in this section can also be embedded into the subposting lists of CS-IIS (i.e., to provide another level of skipping in our approach). Indeed, many of the pruning techniques (as discussed in the next section) that can improve FS can also improve CBR strategies. However, our focus in this article is on conducting CBR efficiently by exploiting that information inherently relevant to the clustering framework itself. Integrating additional pruning techniques to typical- and incremental CBR are beyond the scope of this study and left as future work.

7. RELATED WORK

7.1 Optimization Techniques for FS

There are various optimization techniques used for inverted index searches [Brown 1995; Buckley and Lewit 1985; Moffat and Zobel 1996; Persin 1994; Persin et al. 1996; Anh and Moffat 2005b, 2006, 2001; Lester et al. 2005]. These techniques aim to use only the most promising parts of posting lists and try to increase the efficiency of query processing without deteriorating retrieval effectiveness. For instance, quit- and continue techniques enforce a limit on the number of accumulator entries that can be updated during query evaluation. In this case, memory consumption is reduced, as the accumulators for storing partial similarities can be implemented by dynamic data structures instead of a collection-size array. Furthermore, these two strategies coupled with a skipping index are shown to improve Boolean- and ranking-query efficiency [Moffat and Zobel 1996]. Persin et al. propose to use frequency-sorted indexes to avoid reading entire posting lists from the disk [Persin et al. 1996]. More recently, Anh et al. introduced impact-sorted lists to improve the efficiency of FS [Anh and Moffat 2006, 2001].

Using skip elements in an inverted file to improve query evaluation efficiency in a nonclustering environment was first proposed in Moffat and Zobel [1996] (as discussed in depth in the previous section). Our cluster-skipping inverted file proposed in this article is inspired by this former work, but extends it in various ways. Essentially, it is introduced for use in a new CBR strategy by embedding cluster membership and centroid information into the posting lists.

7.2 Other CBR Strategies

In a hierarchical clustering setup [Voorhees 1986a, 1986b] a CBR system requires several files: the representation of the cluster hierarchy, the centroid vectors, and document vectors. In this setup, a top-down search begins by placing the root of the cluster hierarchy into a max heap [Witten et al. 1994]. During the search, the top element of the heap (which has the highest similarity to the query) is extracted. If it is a document, it is added to the output set. If the extracted element is a cluster, then its children (which may be other clusters or documents) are inserted into the heap according to their similarity to the query (only those with nonzero similarity are considered). The top-down search ends when the heap is empty or a predefined number of documents is retrieved. Notice that the centroid and document vectors are employed during query-cluster- and query-document similarity computations. This is different from our approach, as we employ an inverted index for query processing. Although that earlier work also interleaves query-cluster- and query-document matching, it cannot be called "incremental" in the way we define in this article. In our incremental-CBR strategy, the best clusters are revised after each query term is processed, and the partial similarities of the documents only in these best clusters are updated. The final score of a document depends on whether it has ever been contained in a best-cluster set during the search. The documents in the query output are not determined until all terms are processed.

A bottom-up search strategy, again for hierarchical environments, starts with the top-ranking document(s) (that are at the bottom of the cluster tree) and goes up looking for proper clusters. This approach needs the top-ranking document(s) information, which can only be obtained by a full search (the study also introduces another method which uses the centroids of bottom-most clusters) [Croft 1980]. The search may switch back and forth between documents and clusters. In our case, the set of best-matching clusters is obtained incrementally and the search is always from clusters to documents.

8. CONCLUSIONS AND FUTURE WORK

Cluster-based retrieval (CBR) is a long-studied research area for improving efficiency and effectiveness of document retrieval. Although no conclusive results could be obtained in the past, several researchers reported promising findings for CBR performance in terms of effectiveness and efficiency. Recently, very large document clusters (categorizations) that are obtained either automatically or manually, such as Web directories or digital libraries, have begun to emerge on the Web. This calls for devising efficient CBR techniques.

We introduce an incremental-CBR strategy and a new cluster-skipping inverted index structure for ranking-queries. The new file organization incorporates cluster membership and centroid information along with the usual document information into a single inverted index. In the incremental-CBR strategy, for each query term, the computations required for selecting the best-matching clusters, as well as the best-matching documents of such clusters, are performed in an interleaved manner. The proposed strategy is essentially introduced for providing efficient CBR in compressed environments. We adapt multiple posting-list compression parameters and a cluster-based document id reassignment technique that best fits the features of CS-IIS.

In the experiments, we use various collections and multiple TREC query sets. These datasets constitute the largest collections used for document clustering and CBR. The experiments show that the incremental-CBR strategy with CS-IIS provides significant efficiency improvements while yielding comparable (or sometimes better) effectiveness figures. Our CPU query-processing-time efficiency gains with respect to FS are impressive: up to 45% for Web-style queries. The increment in the size of compressed posting lists is marginal. This overhead can be well compensated for, by the speed of a typical disk, if the index files have to be kept on the secondary storage. In this case our approach leads to another significant advantage: For the first time in the literature, CBR achieves the same number of direct disk accesses as FS, namely, only one access per query term. Furthermore, if we assume that posting lists are kept in the main memory, which is the case for some Web search engines, the reported in-memory gains reflect overall improvements. The experimental results demonstrate the scalability and robustness of our approach.

The future research possibilities, among others, include the following. In this article we concentrated on a term-at-a-time query processing mode. It is also possible to use another efficient alternative, document-at-a-time processing mode, along with the proposed strategy. The proposed skip structure provides interesting data fusion [Nuray and Can 2006] opportunities (i.e., merging FS and CBR results), since both of these processes can be carried out at the same time. Another interesting direction can be that of making the proposed system adaptive to query characteristics; during query evaluation, the number of best clusters to be selected and the centroid-term weighting schemes can be determined according to query length or the weight distributions of query terms. Clearly, updating our data structure is an interesting challenge. We can apply a "distributed free space" technique for future additions to posting lists. Then, given an incremental clustering algorithm (e.g., the incremental version of C^3M [Can 1993]), the complexity of updating CS-IIS is not much higher than that of a typical IIS update. Yet another possible direction for improving storage and efficiency can be using skips only in longer lists, and not in lists of only a few words. Finally, the caching of posting lists constitutes another topic that currently demands serious attention [Baeza-Yates et al. 2007] and can be investigated in our framework, as well.

ACKNOWLEDGMENTS

We thank J. M. Patton for his help in our statistical tests and O. Caki for implementing the compression algorithms. We also thank anonymous referees for their comments.

REFERENCES

Altingovde, I. S., Can, F., and Ulusoy, Ö. 2006. Algorithms for within-cluster searches using inverted files. In *Proceedings of the International Symposium on Computer and Information Sciences (ISCIS)*. 707–716.

Altingovde, I. S., Özcan, R., Öcalan, H. C., Can, F., and Ulusoy, Ö. 2007. Large-Scale cluster-based retrieval experiments on Turkish texts. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retreival*. ACM Press, 891–892.

Anh, N. V., Kretser, O. de, and Moffat, A. 2001. Vector-Space ranking with effective early termination. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retreival*. ACM Press, 35–42

Anh, V. N. and Moffat, A. 2005a. Inverted index compression using word-aligned binary codes. *Inf. Retr. 8*, 1, 151–166.

Anh, V. N. and Moffat, A. 2005b. Simplified similarity scoring using term ranks. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retreival*. ACM Press, 226–233.

Anh, V. N. and Moffat, A. 2006. Pruned query evaluation using pre-computed impacts. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retreival*. Seattle, WA. ACM Press, 372–379.

Baeza-Yates, R., Gionis, A., Junqueira, F., Murdock, V., Plachouras, V., and Silvestri, F. 2007. The impact of caching on search engines. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retreival*. ACM Press, 183–190.

Blandford, D. and Blelloch, G. 2002. Index compression through document reordering. In *Proceedings of the Data Compression Conference*. IEEE, 342–351.

Brin, S. and Page, L. 1998. The anatomy of a large-scale hypertextual Web search engine. *Comput. Netw. 30*, 1–7, 107–117.

Brown, E. W. 1995. Fast evaluation of structured queries for information retrieval. In *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retreival*. ACM Press, New York, 30–38.

Buckley, C. and Lewit, A. F. 1985. Optimization of inverted vector searches. In *Proceedings of the 8th Annual International ACM SIGIR Conference on Research and Development in Information Retreival*. ACM Press, New York, 97–110.

Buckley C. and Voorhees E. M. 2000. Evaluating evaluation measure stability. In *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retreival*. ACM Press, New York, 33–40.

Cacheda, F. and Baeza-Yates, R. 2004. An optimistic model for searching Web directories. In *Proceedings of the 26th European Conference on IR Research (ECIR)*. 364–377.

Cacheda, F., Carneiro, V., Guerrero, C., and Viña, Á. 2003. Optimization of restricted searches in Web directories using hybrid data structures. In *Proceedings of the 25th European Conference on IR Research (ECIR)*. 436–451.

Cambazoglu, B. B. 2006. Models and algorithms for parallel text retrieval. Ph.D. thesis, Bilkent University, Ankara, Turkey.

Cambazoglu, B. B. and Aykanat, C. 2006. Performance of query processing implementations in ranking-based text retrieval systems using inverted indices. *Inf. Process. Manage. 42*, 4, 875–898.

Can, F. 1993. Incremental clustering for dynamic information processing. *ACM Trans. Inf. Syst. 11*, 2, 143–164.

Can, F. 1994. On the efficiency of best-match cluster searches. *Inf. Process. Manage. 30*, 3, 343–361.

Can, F., Altingovde, I. S., and Demir, E. 2004. Efficiency and effectiveness of query processing in cluster-based retrieval. *Inf. Syst. 29*, 8, 697–717.

Can, F. and Ozkarahan, E. A. 1990. Concepts and effectiveness of the cover-coefficient-based clustering methodology for text databases. *ACM Trans. Database Syst. 15*, 4, 483–517.

Croft, W. B. 1980. A model of cluster searching based on classification. *Inf. Syst. 5*, 3, 189–195.

HARMAN, D. 1992. Ranking algorithms. In *Information Retrieval: Data Structures and Algorithms*, W. B. Frakes and R. Baeza-Yates, eds. Prentice Hall, Englewood Cliffs, NJ, (Chapter 14), 363–392.

JAIN, A. K. AND DUBES, R. C. 1988. *Algorithms for Clustering Data*. Prentice-Hall, Englewood Cliffs, NJ.

JARDINE, N. AND VAN RIJSBERGEN, C. J. 1971. The use of hierarchical clustering in information retrieval. *Inf. Storage Retr. 7*, 217–240.

LESTER, N., MOFFAT, A., WEBBER, W., AND ZOBEL, J. 2005. Space-Limited ranked query evaluation using adaptive pruning. In *Proceedings of the 6th International Conference on Web Information Systems Engineering*, New York, 470–477.

LIU, X. AND CROFT, W. B. 2004. Cluster-Based retrieval using language models. In *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM Press, 186–193.

LONG, X. AND SUEL, T. 2003. Optimized query execution in large search engines with global page ordering. In *Proceedings of the 29th International Conference on Very Large Data Bases (VLDB)*. 129–140.

MOFFAT, A. AND ZOBEL, J. 1996. Self-Indexing inverted files for fast text retrieval. *ACM Trans. Inf. Syst. 14*, 4, 349–379.

MURRAY, D. M. 1972. Document retrieval based on clustered files. Ph.D. thesis, Cornell University. Also Rep. ISR-20. National Science Foundation, National Library of Medicine.

NURAY, R. AND CAN, F. 2006. Automatic ranking of information retrieval systems using data fusion. *Inf. Process. Manage. 42*, 3, 595–614.

PERSIN, M. 1994. Document filtering for fast ranking. In *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, ACM Press, 339–348.

PERSIN, M., ZOBEL, J., AND SACKS-DAVIS, R. 1996. Filtered document retrieval with frequency-sorted indexes. *J. Amer. Soc. Inf. Sci. 47*, 10, 749–764.

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

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

SALTON, G. AND BUCKLEY, C. 1988. Term weighting approaches in automatic text retrieval. *Inf. Process. Manage. 24*, 5, 513-523.

SALTON, G. AND MCGILL, M. J. 1983. *Introduction to Modern Information Retrieval*. McGraw Hill, New York.

SILVESTRI, F., ORLANDO, S., AND PEREGO, R. 2004. Assigning identifiers to documents to enhance the clustering property of fulltext indexes. In *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, ACM Press, 305–312.

SILVESTRI, F. 2007. Sorting out the document identifier assignment problem. In *Proceedings of the 29th European Conference on IR Research (ECIR)*. 101–112.

STANFORD UNIVERSITY. 2007. Webbase project Web site. www-diglib.stanford.edu/~testbed/doc2/WebBase.

STROHMAN, T. AND CROFT, W. B. 2007. Efficient document retrieval in main memory. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM Press, 175–182.

TREC 2006. Trec. Web site. http://trec.nist.gov.

TOMBROS, A. 2002. The effectiveness of query-based hierarchic clustering of documents for information retrieval. Ph.D. thesis, University of Glasgow, Glasgow, UK.

VAN RIJSBERGEN, C. J. 1979. *Information Retrieval*, 2nd ed. Butterworths, London.

VOORHEES, E. M. 1985. Cluster hypothesis revisited. In *Proceedings of the 8th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM Press, 188–196.

VOORHEES, E. M. 1986a. The effectiveness and efficiency of agglomerative hierarchical clustering in document retrieval. Ph.D. thesis, Cornell University, Ithaca, NY.

VOORHEES, E. M. 1986b. The efficiency of inverted index and cluster searches. In *Proceedings of the 9th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM Press, New York. 164–174.

WILLETT, P. 1988. Recent trends in hierarchical document clustering: A critical review. *Inf. Process. Manage. 24*, 5, 577–597.

WITTEN, I. H., MOFFAT, A., AND BELL, T. C. 1994. *Managing Gigabytes Compressing and Indexing Documents and Images*. Van Nostrand Reinhold, New York.

ZETTAIR 2007. The Zettair search engine. http://www.seg.rmit.edu.au/zettair/.

ZOBEL, J. AND MOFFAT, A. 2006. Inverted files for text search engines. *ACM Comput. Surv. 38*, 2, 1–56.

Received February 2006; revised November 2007; accepted December 2007