

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 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, 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 is skipped. The proposed approach is essentially tailored for environments where inverted files are compressed, and provides substantial efficiency improvements while yielding comparable or sometimes better effectiveness figures. Our experiments with various collections show that, the incremental-CBR strategy using compressed cluster-skipping inverted file significantly improves CPU time efficiency regardless of the 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.

1. INTRODUCTION

In an information retrieval (IR) system the ranking-queries, or Web-like queries, are based on a list of terms that describe 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, of which dimensions correspond to the terms in the vocabulary of the collection. The value of a vector entry can be determined

This research is supported by The Scientific and Technical Research Council of Turkey (TÜBİTAK) 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.

Permission to make digital/hard copy of part of this work for personal or classroom use is granted without fee provided that the copies are not made or distributed for profit or commercial advantage, the copyright notice, the title of the publication, and its date of appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee.

© 2001 ACM 1073-0516/01/0300-0034 \$5.00

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 the 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 paper, 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, i.e., documents. This strategy, so-called cluster-based retrieval (CBR) is intended to improve both efficiency and effectiveness of the document retrieval systems [Jardin and Van Rijsbergen 1971; Salton 1975; Salton and McGill 1983; Voorhees 1986b]. It can improve efficiency, as the query-document matches are computed for only those documents that are in the 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 the 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 aspects in the last few decades [Salton 1989; Voorhees 1986b; Liu and Croft 2004]. This inconsistency relatively reduced the interest on CBR and its consideration as an alternative retrieval method to full search. On the other hand, the growth of Web as an enormous digital repository of every kind of media, and essentially 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 browse through the categories and assign a query on a particular category. 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 with again manual and/or automatic classification/clustering of the content. Clearly, CBR, as a model of information retrieval, perfectly fits to the requirements of such environments, given that the suspects on its effectiveness and efficiency are remedied. A recent attempt addressing the effectiveness front is by Liu and Croft [2004], which shows that by using language models CBR effectiveness can be significantly better than what it is assumed to be in the literature. The efficiency of CBR is investigated in this paper.

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: the clusters that are most similar to the submitted query are determined by using cluster centroids; ii) best(-matching) document selection: 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 is a presumed to be 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 are rather long. In contrary, the state-of-the art applications for CBR, such as Web directories or digital libraries, involve collections with large number of documents with respect to number of clusters (or, categories) and attempt to respond 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] (please 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 the documents that have non-zero similarity to the query, i.e., the candidates to be the best documents. Next, 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 the current systems [Cacheda et al. 2003; Cacheda and Baeza-Yates 2004]. In this paper, 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 non-zero similarity to the query (note that, this is nothing but the FS). Since only documents from best clusters are returned, the computations (decoding the 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 latter costs specific to best cluster selection stage.

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

In this paper, 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 with 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 an earlier work [Can et al. 2004, Altingovde et al. 2007], which was inspired from [Moffat and Zobel 1996]. Different from previous studies, centroids are now stored in the original term posting lists, which are 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 the computations required for selecting the best-matching documents of such clusters are performed together in an incremental and interleaved fashion. The query terms are processed in a discrete manner in non-increasing term weight order. That is, we envision a term-at-a-time query processing mode in this paper; 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 re-computed and can dynamically change. In the document matching stage of CBR only the portions of the current query term posting list corresponding to the latest best-matching cluster set are considered. The

rest is skipped, hence is not involved in document matching. During document ranking, only the members of the most recent best-matching clusters with a non-zero 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. That is, 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 the memory is scarce (say, for digital libraries and proprietary organizations) and the 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 the document postings that are not from best clusters, a non-trivial cost. We show that, the most important overhead of CS-IIS, longer posting lists, is reduced to an affordable overhead by our compression heuristics and even with a moderate disk, the gains in efficiency can compensate for the slightly longer disk transfer times (given that number of clusters tend to be much smaller than the number 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, 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 issue queries that are restricted to a certain subset of these categories [Cacheda et al. 2003; Cacheda and Baeza-Yates 2004], in addition to usual keyword searches over the entire collection. Such directories, though first created manually (such as DMOZ), have the potential of further growth by using machine learning methods. Also, clustering can be employed for less constrained collections in association with or independently from supervised categorization. All these environments call for efficient methods for processing queries restricted to 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 paper.

CBR may also prove to be beneficial for presenting the results of the queries. Assuming the results are presented based on their clusters, it is possible for the user to browse a cluster and discover some other similar and potentially relevant documents, which do not capture any of the query words and 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 paper, we envision that CBR is a worthwhile strategy in certain domains, such as those described above, provided that it can be competitive with FS in terms of effectiveness and efficiency. Given the recent promising findings on effectiveness [Liu and Croft 2004], we focus on the latter and aim to show that efficient CBR is an attainable goal, as well.

1.2 Contributions

The contributions of this study are:

- *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 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. By 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, 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 (and typical-CBR, which actually involves FS during best document selection stage) in efficiency while yielding comparable (or sometimes better) effectiveness figures. We also show that efficiency of our approach scale well with the collection size. The proposed approach is also superior to the FS approach 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:* 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 paper 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 the 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, and 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 the Web search engines (such as Google) and IR systems. However, our approach proposed in this paper can be applicable to Boolean queries, as well.

In the vector space model, 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 the several “term weighting” methods described in the literature [Salton and Buckley 1988]. The goal of these methods is to assign higher weights to the terms that can potentially discriminate a document among others, and vice versa. In this study, the document term weights are assigned using the *term frequency (tf) x 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 $\ln(\text{number of all documents/number of documents containing } t) + 1$. 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 following formula

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

The term weights for query terms ($w_{q,t}$) are calculated in a similar fashion to document term weights. In this case, for computing term frequency component, we use augmented normalized frequency formula defined as $(0.5 + 0.5 \times tf / \text{max-tf})$. Here *max-tf* denotes the maximum number of times any term appears in the query vector. The *idf* component

is obtained in exactly the same manner with the document terms. No normalization is done for query terms since it does not affect document ranking.

After obtaining weighted document (d) and query (q) vectors in a T dimensional vector space the query-document matching is performed using the following formula [Salton and McGill 1983].

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

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 every document vector (most of which would probably yield no similarity at all), but only those that have at least one common term with the query. An inverted file has a header part, including list of terms in the collection, and pointers to the posting lists for each term. Along with the terms, f_t , number of documents in which this term appears, is kept. A posting list for a term consists of the 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 the memory (note that variations are possible [Harman, 1992]). The weighted query vector is constructed as described above. 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 term frequency f_t stored in the IIS header. Next, using a similarity function the partial similarity of the query to the 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 accumulator are normalized, i.e., divided by the pre-computed document lengths. Finally, the accumulators (documents) are sorted in descending similarity order and returned as the query output. If only 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; Witten et al. 1994].

In this paper, a ranking-query evaluation as described in the above 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 stop words, as we mention in the experimental set-up).

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 of the researches in the literature focus on the hierarchic methods (see, for example, [Willett 1988]). Cluster-based retrieval is intended to improve both efficiency and effectiveness of the 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, determining the maximum centroid length and 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 hierarchical clustering experiments described by Voorhees [Voorhees 1986a; Voorhees 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 as 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 the shortest centroid vectors that cause minimal deterioration on effectiveness. However, the results reported in that work show variability to draw a conclusion for the relationship of the centroid length and effectiveness for several hierarchical CBR techniques. In a recent work, several methods for centroid generation are reviewed and it is concluded that the need for an extensive investigation of centroids on CBR effectiveness still exists [Tombros 2002].

Table I. Term weighting schemes for centroids

Weighting scheme	Term frequency (<i>tf</i>)	Inverse document frequency (<i>idf</i>)
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	$\ln \frac{\text{sum of occurrence numbers in the clusters}}{\text{number of occurrence in the cluster}} + 1$

In the rest of this paper, we assume that three centroid term weighting schemes are employed: CW1, CW2, and CW3; in all of them the weight of a centroid term is computed by the formula $tf \times idf$. In CW1, tf is taken as 1, in CW2 and CW3 it is taken as the number of occurrence of a term in the cluster, i.e., *within-cluster term frequency*. In CW1 and CW2, idf is taken as $\ln(\text{number of clusters} / \text{number of centroids including the term}) + 1$, in CW3, it is taken as $\ln(\text{sum of occurrence numbers in the centroids} / \text{number of occurrence in the cluster}) + 1$. During the best-cluster selection stage of query processing, weights are normalized by using the pre-computed 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 number of best matching clusters using centroids, ii) selection of d_s number of best matching documents of the selected best matching clusters. For item (i) we have two file structure possibilities: centroid vectors, and IIS of centroids. For item (ii) we again have two possibilities: document vectors, and IIS of all documents. One remaining possibility for (ii), 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 overhead. Hence, possible combinations of (i) and (ii) determine four different implementation alternatives.

In [Can 1994] the efficiency of the above alternatives is measured in terms of CPU time, disk accesses, and storage requirements in a simulated environment defined in [Voorhees 1986b]. It is observed that the alternative 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 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 those documents that are retrieved by the second stage (i.e., FS using the document index) but not belong to the best clusters. A similar approach is typically used for processing queries restricted to certain categories on Web directories (with the only distinction that best cluster(s) are explicitly specified by the user instead of an automatic computation) [Cacheda et al. 2003, Cacheda and Baeza-Yates 2004]. Throughout the paper, we consider this particular implementation as the baseline best-match CBR method and refer to it as *typical-CBR*.

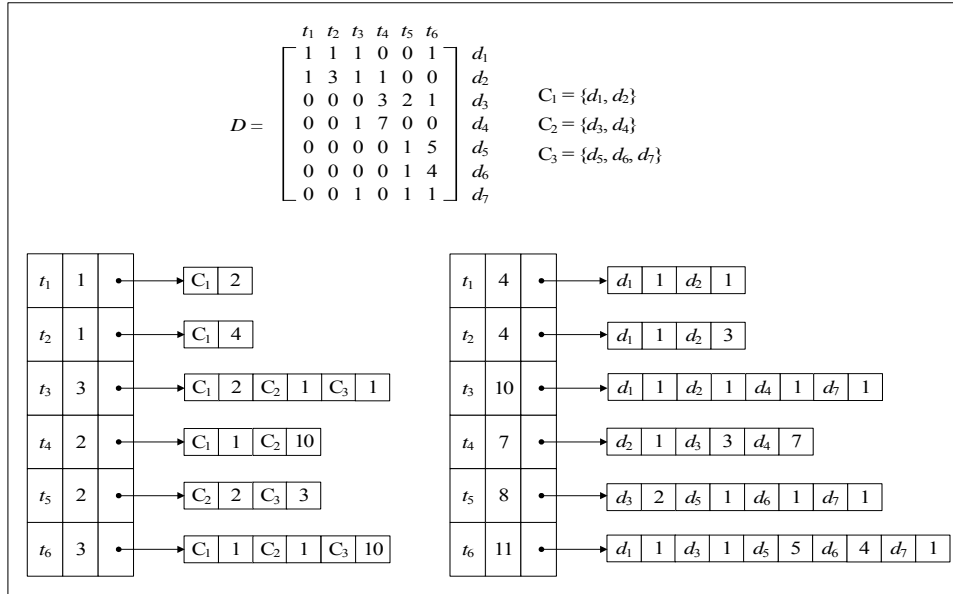


Fig. 1. Centroid and document IIS for typical-CBR.

In Figure 1, we illustrate the centroid and document IIS files for this strategy. The example provided in Figure 1 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. It shows that document 2 (d_2) contains term 1 (t_1) once and t_2 three times. We assume that, for simplicity, all terms appearing in the member documents of a cluster are used in the centroid and 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 and the final result filtering would also incur additional costs. In [Altingovde et al. 2006], efficiency trade-offs for typical-CBR are discussed, but those arguments are essentially for the uncompressed environments and the findings are not directly applicable to our framework here.

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

In this paper, 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 paper is different than 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, and
- 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 posting list for a term t includes the following documents: 3, 7, 11, 14, 21, 24; using d-gaps this 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 the 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 than larger ones. 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; Witten et al. 1994]. More recently, Anh and Moffat [2005a] propose 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., $\langle d, f_{d,r} \rangle$ 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 modeling 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 values. During encoding and decoding, the b value is determined for a particular posting list I_t by the formula:

$$b = 0.69 \times \frac{N}{f_t} \quad (1)$$

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 how the Golomb code with local Bernoulli model can be adapted to cluster-skipping IIS in detail.

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 stores the documents of each cluster in a group adjacent to each other. It contains a skip-element preceding each such group to store the id of 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 it is required in typical-CBR methods.

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., non-increasing) 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 sub-posting list IS_i in I_t
 - i) Access the skip- and centroid-element $\langle Cid, address \rangle$ and $\langle sub\text{-posting list length, average within-cluster term frequency} \rangle$ from IS_i
 - ii) Compute $w_{Cid,t}$ using $sub\text{-posting list length}$ and $average\ within\text{-cluster term frequency}$
 - 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 sub-posting list IS_i in I_t
 - Access the skip-element $\langle Cid, address \rangle$ from IS_i
 - If Cid is in $BestClus$
 - For each document in IS_i
 - Access the typical element $\langle document\ id\ Did, term\ frequency\ tf \rangle$
 - Compute $w_{Did,t}$ using tf_i
 - $DAcc[Did] \leftarrow DAcc[Did] + w_{q,t} * w_{Did,t}$
 - 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.

In Figure 2, the posting list for t_6 includes documents from three clusters. For the first two clusters, the centroid-elements simply store $\langle 1, 1 \rangle$ 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 $\langle 3, 3 \rangle$ since there are three documents in cluster (d_5, d_6, d_7) and 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 there is 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 that 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 sub-posting list (i.e., *sub-posting list length* x *average within-cluster term frequency*), as shown in the 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 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 that cluster lengths (i.e., centroid normalization factors used in matching [Salton and Buckley 1988]) are pre-computed and stored just like document lengths for whichever term weighting scheme is used. During query processing, centroid term weights are normalized by using the pre-computed 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 progress both processes in incremental fashion. In this new strategy, the query terms are processed in decreasing order according to their weights. 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 are available in the skip- and centroid-elements (as mentioned in the above section), so during the first pass it is sufficient to access *only* to those elements of each sub-posting list. In the second pass, only those documents whose clusters fall into the *best-clusters-so-far* are considered, while the system skips the documents from the non best-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.

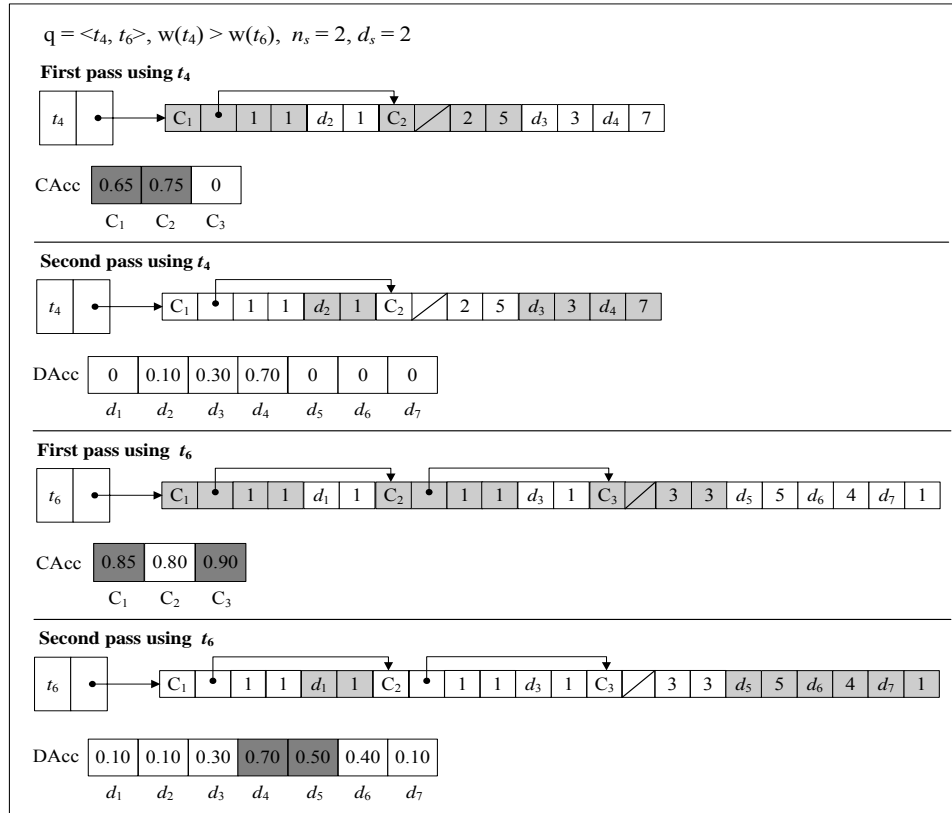


Fig. 4. Example query processing using incremental-CBR strategy (accessed and decompressed list elements are shown with light gray, best documents and clusters are shown with dark gray).

For instance, assume a query that contains the terms $\{t_4, t_6\}$ and 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 *best-clusters-so-far*, and in the second pass the document accumulator entries for the documents in these clusters, namely, 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 cluster accumulator entries for clusters C_1, C_2 and C_3 with the additional values 0.20, 0.05 and 0.90, respectively. Now, the *best-clusters-so-far* includes C_1 and C_3 with scores 0.85 and 0.90 whereas C_2 with score 0.80 is out, and thus the documents from these two clusters are considered but sub-posting list for C_2 is skipped during the second

pass. That is, 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, the proposed incremental-CBR strategy with the CS-IIS file has two major advantages: First, embedding cluster information into the IIS and the incremental query evaluation method eliminate the need for a separate centroid IIS and hence disk access time to retrieve its posting lists. This means, in a memory-scarce environment where the index files are kept on disk, incremental-CBR achieves half of the number of direct disk accesses required by typical-CBR, and the same number of direct disk accesses required by FS. Second, cluster skipping and thus, decoding only relevant portions of 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, 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, the 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 “sub-posting list length – average within-cluster term frequency,” and iii) the typical elements of type “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 sub-posting 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, i.e., 1.

There are two subtle issues regarding the above 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 sub-posting lists) and 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 is uniformly distributed among the 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).

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

The second important observation from the above representation is that, for the cluster-skipping IIS, the first document id in *each* sub-posting list per cluster (e.g., d_1 , d_4 and d_7 in the above 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 document ids in a posting list as close as possible, so that the frequency of small d-gaps improves compression rates [Silvestri et al. 2004]. In that 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 below. 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, the 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 the documents in a cluster is determined by the order of entrance of these documents into the cluster. Notice that, a similar approach using 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 two fold: i) in each sub-posting 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 sub-posting list can be reassigned a smaller value. Indeed, with a little main-

memory consumption, it is possible to amplify the benefit mentioned in (ii) significantly. In each cluster, documents are assigned a *real id*, which is determined as described above, and a *virtual id*, which starts from 1 and increments by 1, just to be used for the compression purposes. During compression, virtual ids are compressed, so that each sub-posting list would start with a considerably smaller id than the original one. During query processing, an array is kept in main memory to store prefix sum of cluster sizes, 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 sub-posting list) in the size-sum array to obtain the real id, and 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 documents. The documents from C_1 and C_2 will be assigned to real id's 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 sub-posting list is decoded as 2, it will be added to size-sum array value for C_2 , which is also 2, to obtain the real id as 4.

Note that, number of clusters would be smaller than the number of documents in the order of magnitudes, so that storing size-sum array in the 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, i.e., 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 Eq. 3, since the virtual documents ids in each sub-posting list can range from 1 to "average cluster size" on the average.

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

As another alternative, we can define a dedicated b value to compress each sub-posting list separately (Eq. 4). Note that, cluster size C_i can be easily computed from the size-sum array as the difference of 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., sub-posting list length) and will be decoded immediately before the decoding of d-gaps start. In Section 6, we evaluate and compare the storage figures for various compression schemes and parameters.

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

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	564 MB (text)	210,158	229,748	1,640	29,545,234	128
AQUAINT	3 GB (text)	1,033,461	776,820	5,163	170,004,786	200
WEBBASE	140 GB (HTML)	4,293,638	4,290,816	13,742	790,291,775	312

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, are 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 2007]. This latter dataset, referred to as the WEBBASE, includes pages collected from the US government Web sites during the first quarter of 2007. As there are no query topics and relevance judgments for this dataset, it is solely used for evaluating query processing efficiency. During the indexing stage, we eliminated English stop-words, and indexed the remaining words, and no stemming is used. For the WEBBASE dataset, the 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 140 GB 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 27 GB.

The datasets are clustered using C³M algorithm [Can and Ozkarahan 1990] in partitioning mode, which yields 1,640, 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 has been reported that the effectiveness increases up to a certain n_s value, 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 around 10 to 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, 516 and 1374 for the corresponding datasets). In this paper, we provide results for retrieving top-1000 documents, i.e., number of best documents to be selected, d_s , is 1000.

The clustering of the largest dataset (WEBBASE) takes around twelve hours using a rather out-dated implementation of C³M algorithm. Once the clustering is completed, creating the typical IIS and CS-IIS takes almost equal times, which is around 6 hours for this dataset, by again using an unoptimized implementation. Nevertheless, any partitioning type clustering algorithm could be used in our setup, given that the algorithm can provide reasonable effectiveness by accessing a relatively small percentage of all clusters.

5.2 Query Sets and Query Matching

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, Q_{set1}	47	31.8	Q_{short}	2.5	1	4
			Q_{medium}	10.8	4	30
FT, Q_{set2}	49	38.1	Q_{short}	2.4	1	3
			Q_{medium}	8.2	2	19
			Q_{long}	190.0	13	612
FT, Q_{set3}	49	33.4	Q_{short}	2.4	1	3
			Q_{medium}	7.3	3	19
AQUAINT, Q_{set1}	50	131.2	Q_{short}	2.5	1	4
			Q_{medium}	9.4	4	20
WEBBASE, Q_{set1}	50,000	N/A	Q_{short}	2.3	1	9

For the FT dataset, we used three different query sets along with their relevance judgments that are obtained from the TREC Web site [TREC 2006]. The three query sets, referred to as Q_{set1} , Q_{set2} and Q_{set3} , 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 that are from datasets other than the ones used in this paper. Such irrelevant judgments are eliminated, and for each query set we produce a relevance judgment file, which includes only the documents from the FT dataset. A few of the queries do not have any relevant documents, and they are 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 TREC 2005 robust track. Finally, for the WEBBASE dataset, the efficiency task topics of TREC 2005 terabyte track are employed. Note that, this query set have been 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 that this query set is 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 above. *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, *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 similar documents to a particular document and the document itself serves as a query. This provides insight on the behavior of retrieval system at extreme conditions. Table III provides 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 that the effectiveness does not vary significantly for centroid lengths 250, 500 and 1000; whereas using all cluster terms in the centroid yields slightly better performance than those earlier ones. 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 us 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 as 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 the query processing weights are normalized by using the pre-computed 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 site¹. Implementations of the IR strategies are tuned to optimize 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 works [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, i.e., more than one term's posting list is not memory resident simultaneously. The document lengths and cluster lengths are pre-computed.

In what follows, we first compare the effectiveness figures of the incremental-CBR strategy with those of the 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 that incremental-CBR is better than FS in total query processing performance (involving in-memory evaluation and disk accesses) with a reasonable overhead in the storage requirements. Finally, we show that incremental-CBR is superior than not only a basic implementation of FS but 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 result files corresponding to the above table are available at our Web site¹.

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 by using a publicly available search engine, Zettair [Zettair 2007], 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 our own implementations.

¹ <http://www.cs.bilkent.edu.tr/~ismaia/incrementalCBR.htm>

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 site¹.

Table IV. MAP values for retrieval strategies ($n_s=164$ for FT, $n_s=516$ for AQUAINT, $d_s=1000$)

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

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 above results it is clear that the proposed strategy has no adverse affect on CBR effectiveness and in particular cases, it can even improve effectiveness. In particular, Table IV reveals that incremental-CBR strategy is better than the typical-CBR for the majority of the cases, although the absolute MAP 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 as good. For this purpose, we examine

the performance differences of these two approaches 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 6 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. So a significant result would be a p-value that is 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 the average, FS outperforms each cluster method in terms of precision. However, the only significant difference (based on p-values) is the difference between CW3 for typical-CBR and FS ($p < 0.01$). In this case, FS significantly outperforms typical-CBR with CW3. However, in the other tests there is a lack of evidence that FS significantly outperforms CBR. Since CBR with CW1 and CW2 outperform 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 effectiveness of CBR, but it aims to improve efficiency without deteriorating the effectiveness of the typical-CBR while providing compatible effectiveness with FS. Recall that, there are 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. 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 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 the compressed file sizes for the evaluated IR strategies. In particular, the term frequency values in typical IIS and 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 to be unaffected from the document id reassignment methods for typical IIS.

Table V. IIS file sizes (in MBs) for FS and Incremental-CBR (LB: local Bernoulli model, OrgID: original doc ids, ReID: reassigned doc ids)

Dataset	FS					Incremental-CBR (CW2-3)				
	Uncomp. IIS file size	Golomb (LB)		Elias- γ		Uncomp. CS-IIS file size	Golomb (LB)		Elias- γ	
		OrgID	ReID	OrgID	ReID		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)

Table V reveals that for the 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 vs. 1,076 MB). We attribute this 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 Belloch 2002]. This phenomenon is strongly emphasized by the experiments with the reassigned document ids and further discussed below. Nevertheless, for the WEBBASE dataset, the typical IIS size drops from 6,322 MB to 1,076 MB (17%) and 767 MB (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 (27 GB) for respective cases. This conforms to the results reported in other works in the literature [Witten et al. 1994]. On the other hand, it is seen that the compression gains on CS-IIS by using original document ids are not as good, and for 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 been exploited yet.

Next, we applied the document id reassignment method mentioned in Section 4.2, so that 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, to reflect the distribution of sub-posting lists as accurate 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 WEBBASE dataset drops from 2,315 to 968 MB, a reduction of more than 50%. This is even less than the compressed size of typical IIS (1,079 MB) for the corresponding case. As it is mentioned before, the insensitivity of typical IIS for reassigned ids is caused from the characteristics of the Golomb code, which cannot exploit the locality (i.e., it should still use the same b parameter for LB after reassignment). In particular, for FT and AQUAINT datasets the reductions in the compressed index sizes are at most 3%, hard to call as an improvement. For WEBBASE, there is even a slight increase (0.3%) in the 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 the 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 the gains are still less impressive in comparison to CS-IIS. Noticeably, the storage space used for compressed index files of FT and AQUAINT drops by 2% and 9%, respectively. For WEBBASE, there is no improvement on the index size, but again a slight increase is observed. This is due to the fact that, the originally URL-ordered ids for this dataset provides quite strong locality, and the reassignment based on clustering does not further improves 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 1,473 MB 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 would provide better compression rates (as in the cases of FT and AQUAINT) with respect to Elias- γ , ii) the Golomb code is rather not sensitive to any locality (the file sizes for random and URL-sorted experiments are very close, 1,132 and 1,076 MB, respectively) whereas Elias- γ is just the reverse (i.e., the index size drops from 1,473 to 776 MB), 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 844 MB on disk, which is only 10%

larger than the typical IIS for corresponding case (770 MB). This is a striking result for the space utilization of CS-IIS that is obtained by using a cluster-based document id reassignment technique which 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 sub-posting lists, to encode the first document of each sub-posting 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 documents ids. In this case, the first document id of each posting list, which should be compressed as-is, takes 330 MB and 232 MB of the resulting CS-IIS file, for the Elias- γ and the 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 first document ids in sub-posting lists). In this case, only 100 MB and 76 MB of CS-IIS is devoted to first ids, again for the Elias- γ and the 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 much better compression ratio with respect to Elias- γ , i.e., leading to a further 24% reduction in size. This experiment shows that, efficient compression of the first document id in each sub-posting list of CS-IIS is important for the overall compression efficiency, and the heuristic outlined in this paper 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 the 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 the Golomb coding with the LB model and Elias- γ schemes prove to be 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 that are obtained by the id reassignment and the Elias- γ encoding for d-gaps i.e., those shown as bold in Table V.

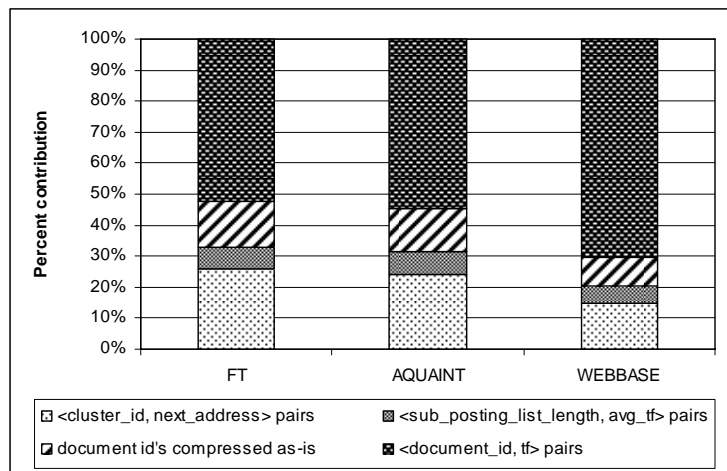


Fig. 5. Contribution of CS-IIS posting list elements to compressed file sizes for the three datasets.

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 actual $\langle \text{document id, tf} \rangle$ pairs, whereas 15% is used for the skip-elements (i.e., in the form of $\langle \text{cluster id, next address} \rangle$), and 5% is used for the centroid-elements (i.e., in the form of $\langle \text{sub-posting list length, avg. tf} \rangle$). Since each sub-posting 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 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 the additional storage in CS-ISS reduces as the dataset gets larger, i.e. 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 above.

Finally note that, the compression process takes the same time for corresponding cases by using our own implementations, ranging from a few minutes (for FT) to an hour (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 that of CW1.

Table VI. Efficiency comparison of FS and Incremental-CBR (times in ms)

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

The results reveal that incremental-CBR decompresses significantly smaller number of elements compared to FS. This is caused by the fact that the former decompresses only relevant portions of a posting list, whereas FS, of course, must decode the entire posting list for a query term (note that, 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 the incremental-CBR in terms of number of decode operations are more emphasized, ranging from 58% to 80% of the decode operations by FS. For CW2, incremental-CBR decodes more elements, but still the number of decoded elements is almost half of the FS case. These savings are reflected to time figures rather conservatively, especially for shorter queries. The time savings improve as the queries become longer (e.g., for AQUAINT 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.

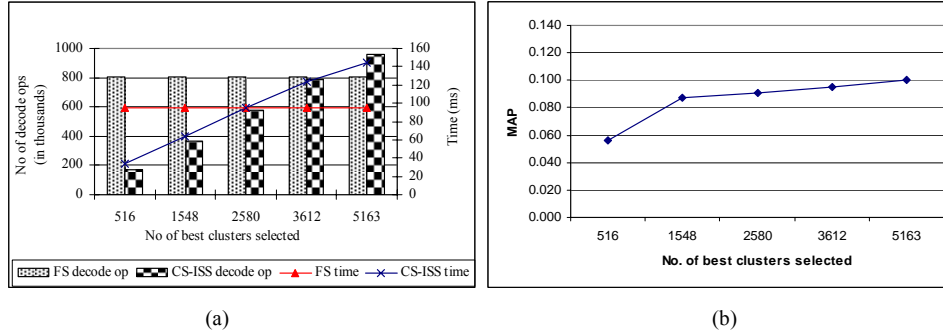


Fig. 6. Effects of the selected best cluster number on (a) processing time and decode operation number, (b) effectiveness (for Q_{medium} using CW1 on AQUAINT dataset).

In Figure 6.a, we plot the number of best clusters selected vs. average number of decode operations (shown on the left y-axis of the plot) and average CPU query processing time (shown on the right y-axis of the plot) for FS and incremental-CBR, for Q_{medium} using CW1 centroid weighting scheme and the AQUAINT dataset. At the extreme point, all clusters are selected and incremental-CBR degenerates into FS. The number of decode operations realized by incremental-CBR and execution time is lower than that by FS until more than 50% of clusters (i.e., greater than 2580) are selected. Nevertheless, in practical CBR systems, the number of best clusters to be selected is a relatively small percentage of the total number of clusters [Can and Ozkarahan 1990; Salton 1975].

In Figure 6.b, we plot the variation of the number of best clusters selected vs. effectiveness. Note that, after 30% of clusters are selected as best clusters, the MAP figures change slightly. Thus, for AQUAINT dataset it is possible to set best clusters as 30% of all clusters (i.e., 1548). Note that, even for this case, both the number of decompression operations and execution time are still significantly less than those for FS (see Figure 6.a). For the sake of uniformity, we keep 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, the 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 the 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, Q_{set2}	Q_{short}	10.54	12.56	19
	Q_{medium}	51.09	60.78	19
	Q_{long}	1670.77	1962.12	17
AQUAINT, Q_{set1}	Q_{short}	73.48	83.55	14
	Q_{medium}	345.64	391.26	13
WEBBASE, Q_{set1}	Q_{short}	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 10 MB/s. In this case, the additional sequential read time cost of CS-IIS with respect to FS for processing a query in Q_{short} set of WEBBASE would be around only ≈ 1 ms (i.e., $(157.75-147.96)\text{KB} / 10 \text{ MB/s}$). For this latter case, FS takes 66 ms in CPU whereas incremental CBR takes 36 and 57 ms for CW1 and CW2 cases, respectively (see Table VI). Clearly, even with a slow disk, in-memory time improvements are far larger than the disk read overhead (i.e., 30 ms (for CW1) and 9 ms (for CW2) vs. 1 ms). 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 become the conclusive improvements.

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 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 it and compare 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 pointer to the next skip-element (and block). This was shown to be very efficient for conjunctive Boolean queries in a compressed environment. In particular, after the first posting list is processed, a candidate set of document id's are obtained, which are looked for in the other lists.

Obviously, while 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 one, 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 allow the skipping inverted index to be used. Since the effectiveness figures of continue is quite close to the FS without any pruning (referred to as *typical FS* below), we prefer to use the continue strategy in this paper.

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, the non-zero accumulators that are 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 non-zero accumulators. We refer to this strategy as *skipping FS*.

In [Moffat and Zobel 1996], each posting list can have different number of skip elements, according to the size of the posting list and the 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 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 around 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 279 MB, which is 18% larger than the IIS with no skips (i.e., 236 MB, as shown in Table V). In Figure 7.a, the MAP figures using this IIS file and varying number of accumulators (k) is shown. As expected, the effectiveness figures at $k=10K$ is as good as the effectiveness score when all 1M accumulators are available, i.e., typical FS.

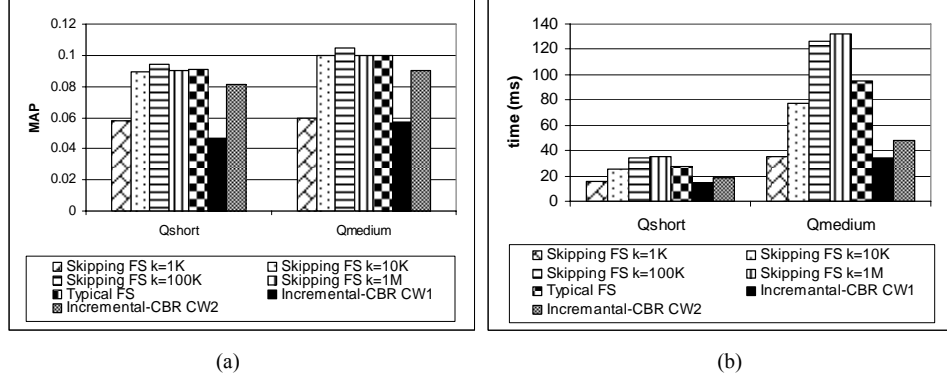


Fig. 7. (a) Effectiveness of skipping FS vs. no of accumulators, (b) Query processing time of IR strategies.

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
<i>Qshort</i>	40,377	106,150	190,556	190,635	162,824	37,860	73,249
<i>Qmedium</i>	123,777	408,601	886,858	930,714	802,740	172,415	313,291

In Figure 7.b, CPU execution times for skipping FS strategy with 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 comparison). Clearly, skipping FS improves time performance of typical FS, up to 41% for *Qshort* and 63% for *Qmedium* when $k = 1K$. However, for this case, MAP scores also decrease. For $k = 10K$ 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) still remains to be superior. In Table VIII, we report the number of decompression operations for the corresponding cases. Again, skipping FS improves over typical-FS, but cannot catch the incremental-CBR, for $k = 10K$ case.

Finally note that, dynamic pruning techniques such as the one described above 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 skipping FS approach. Similarly, the skipping strategy in this section can also be embedded into the sub-posting lists of CS-IIS (i.e., to provide another level of skipping in our approach). That is, many pruning techniques (as discussed in the next section) that can improve FS can also improve the CBR strategies. However, our focus in this paper is conducting CBR efficiently by exploiting the information that is 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 2001, 2005b, 2006; Lester et al. 2005]. These techniques aim to use only the most promising parts of posting lists and try to increase 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 1996]. More recently, Anh et al. introduced impact-sorted lists to improve the efficiency of FS [Anh and Moffat 2001, 2006].

Using skip-elements in an inverted file to improve query evaluation efficiency in a non-clustering 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 paper 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; Voorhees 1986b], a CBR system requires several files: the representation of the cluster hierarchy, the centroid vectors and the document vectors. In this set-up, 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 non-zero similarity, are considered). The top-down search ends when the heap is empty or a pre-defined number of documents is

retrieved. Notice that, the centroid and document vectors are employed during the query-cluster and query-document similarity computations. This is different from our approach as we employ an inverted index to 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 paper. 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) which is 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 are obtained incrementally and 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, has 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 and selecting 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 and up to 45% for Web style queries. The increment in the size of compressed posting lists is marginal. This overhead can be well-compensated 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, i.e., 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 paper, we concentrated on 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 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 the query length or the weight distributions of the 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 the complexity of a typical IIS update. Yet another possible direction for improving storage and efficiency can be using skips in only “longer lists” but not in the lists of only a few words. Finally, the caching of posting lists is another topic that currently takes serious attention [Baeza-Yates et al. 2007] and can be investigated in our framework, as well.

ACKNOWLEDGEMENTS

We thank Jon M. Patton for his help in our statistical tests and Oguzhan 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 Proc. of ISICIS'06, pp. 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. ACM, pp. 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. ACM, pp.35-42
- ANH, V.N. AND MOFFAT, A. 2005a. Inverted index compression using word-aligned binary codes. Information Retrieval 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, ACM, pp 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, Seattle, Washington, USA. SIGIR '06. ACM, pp. 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. ACM, pp. 183-190.
- BLANDFORD, D. AND BLELLOCH, G. 2002. Index compression through document reordering. In Proceedings of the Data Compression Conference. IEEE, pp. 342-351.
- BRIN, S. AND PAGE, L. 1998. The anatomy of a large-scale hypertextual Web search engine. Computer Networks 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, ACM, New York, pp. 30-38.
- BUCKLEY, C. AND LEWIS, A. F. 1985. Optimization of inverted vector searches. In Proceedings of the 8th Annual International ACM SIGIR Conference, ACM, New York, pp. 97-110.
- BUCKLEY C. AND VOORHEES E.M. 2000. Evaluating evaluation measure stability. In Proceedings of the 23rd Annual International ACM SIGIR Conference, ACM, New York, pp. 33-40.
- CACHEDA, F. AND BAEZA-YATES, R. 2004. An optimistic model for searching Web directories. In Proc. of 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 Proc. of the 25th European Conference on IR Research (ECIR), pp. 436–451.
- CAMBAZOGLU, B.B. 2006. Models and algorithms for parallel text retrieval. PhD. 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. Information Processing and Management 42, 4, 875-898.
- CAN, F. 1993. Incremental clustering for dynamic information processing. ACM Transactions on Information Systems 11, 2, 143-164.

- CAN, F. 1994. On the efficiency of best-match cluster searches. *Information Processing and Management* 30, 3, 343-361.
- CAN, F., ALTINGOVDE, I.S., AND DEMIR, E. 2004. Efficiency and effectiveness of query processing in cluster-based retrieval. *Information Systems* 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 Transactions on Database Systems* 15, 4, 483-517.
- CROFT, W. B. 1980. A model of cluster searching based on classification. *Information Systems* 5, 3, 189-195.
- HARMAN, D. 1992. Ranking algorithms. In *Information retrieval: data structures & algorithms* (Chapter 14) edited by W.B. Frakes, R. Baeza-Yates, Prentice Hall, Englewood Cliffs, NJ, 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. *Information Storage and Retrieval* 7, 217-240.
- LESTER, N., MOFFAT, A., WEBBER, W., AND ZOBEL, J. 2005. Space-limited ranked queryevaluation using adaptive pruning. In *Proc. 6th International Conference on Web Information Systems Engineering*, New York, pp 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*, ACM, pp. 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)*, pp. 129-140.
- MOFFAT, A. AND ZOBEL, J. 1996. Self-indexing inverted files for fast text retrieval. *ACM Transactions on Information Systems* 14, 4, 349-379.
- MURRAY, D. M. 1972. Document retrieval based on clustered files. PhD. Thesis, Cornell University. Also Report ISR-20 to National Science Foundation and to the National Library of Medicine.
- NURAY, R. AND CAN, F. 2006. Automatic ranking of information retrieval systems using data fusion. *Information Processing and Management* 42, 3, 595-614.
- PERSIN, M. 1994. Document filtering for fast ranking. In *Proceedings of the 17th ACM SIGIR Conference*, ACM, pp. 339-348.
- PERSIN, M., ZOBEL, J., AND SACKS-DAVIS, R. 1996. Filtered document retrieval with frequency-sorted indexes. *Journal of the American Society for Information Science* 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. *Information Processing and Management* 24, 5, 513-523.
- SALTON, G. AND MCGILL, M. J. 1983. *Introduction to Modern Information Retrieval*. McGraw Hill, New York, NY.
- 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*, ACM, pp. 305-312.
- SILVESTRI, F. 2007. Sorting out the document identifier assignment problem. In *Proc. of 29th European Conference on IR Research (ECIR)*, pp. 101-112.
- STANFORD University WebBase Project Web Site, 2007. 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, ACM, pp. 175-182.

TREC Web site, 2006. <http://trec.nist.gov>.

TOMBROS, A. 2002. The effectiveness of query-based hierarchic clustering of documents for information retrieval. PhD. 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, ACM, pp. 188-196.

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

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

WILLETT, P. 1988. Recent trends in hierarchical document clustering: A critical review. Information Processing and Management 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, available at <http://www.seg.rmit.edu.au/zettair/>.

ZOBEL, J. AND MOFFAT, A. 2006. Inverted files for text search engines. ACM Computing Surveys 38, 2, 1-56.

Received February 2006; revised June 2007, November 2007; accepted December 2007.