# Algorithms for Within-Cluster Searches Using Inverted Files

Ismail Sengor Altingovde, Fazli Can, and Özgür Ulusoy

Department of Computer Engineering, Bilkent University
06800, Ankara, Turkey
{ismaila, canf, oulusoy}@cs.bilkent.edu.tr

**Abstract.** Information retrieval over clustered document collections has two successive stages: first identifying the best-clusters and then the best-documents in these clusters that are most similar to the user query. In this paper, we assume that an inverted file over the entire document collection is used for the latter stage. We propose and evaluate algorithms for within-cluster searches, i.e., to integrate the best-clusters with the best-documents to obtain the final output including the highest ranked documents only from the best-clusters. Our experiments on a TREC collection including 210,158 documents with several query sets show that an appropriately selected integration algorithm based on the query length and system resources can significantly improve the query evaluation efficiency.

## 1 Introduction

Document clustering is one of the earliest approaches proposed for improving effectiveness and efficiency of information retrieval (IR) systems [8, 9]. With the fast growth of the Web, the amount of digital text data has also increased enormously in the last decade. This has given rise to a new interest on document clustering, not only due to the historically known promises of effectiveness and efficiency improvements, but also for several other purposes and application areas such as data visualization or preprocessing for several data mining applications. Moreover, the advent of the Web also fueled creation of some of the largest document hierarchies of the digital age. As an alternative to search engines that index all the terms on the Web pages and provide keyword-based searches, Web directories (such as Yahoo! and DMOZ) exist and attract attention of users. Such Web directories are formed by manually assigning Web pages to the categories of a topic hierarchy by human experts.

A user accessing an IR system with document clusters has three possible methods for satisfying his/her information needs: browsing, keyword-based (ad hoc) querying, or browse-based querying (i.e., the user can first browse through the categories until (s)he reaches to the cluster(s) (s)he is interested in and then pose a keyword-based query to be evaluated under this particular cluster(s)). Notice that, for any given IR system involving document clusters (or categories) -either created automatically or manually, for legacy data or Web documents and in a flat or hierarchical structure- the retrieval process involves two basic steps: finding the relevant clusters that best match to the user query and identifying the most relevant documents within these clusters. The first step, best(-matching) cluster selection, can be achieved either automatically

A. Levi et al. (Eds.): ISCIS 2006, LNCS 4263, pp. 707 – 716, 2006.

by matching a user query to cluster representatives (centroids), or manually, as provided by user browsing. The second step, best(-matching) document selection, usually involves an inverted index structure of entire document collection, which is employed during the query-document matching. Thus, the result of best-document selection step is a ranked list of all documents in the collection according to their relevance to the query. The best-clusters obtained in the first step is then used to filter the best-documents in the second step, so that only those documents that have the highest similarity to the query and come from the best-clusters remain in the final output set. This integration step of best-clusters and best-documents can be postponed to the point where both sets are separately identified, or can be somehow embedded to best-document selection process. In this paper, we try to figure out how and when the best-cluster set information should be *integrated* while selecting best-documents with a traditional inverted index based query evaluation strategy. Interestingly, although the history of document clustering spans a few decades, the above question at a detail level of practical implementation has only attracted attention very recently and has been discussed in a few research studies [1, 2, 3, 4, 5].

The main contribution of this paper is proposing and evaluating result integration algorithms to identify a query's best-matching documents that lie within a pre-determined set of best-clusters. It is assumed that best-document selection process involves an inverted index structure (IIS) over all documents, which is the case in most practical systems, and thus such an integration step will be required. The integration algorithm would be of critical value for improving query processing efficiency for both traditional IR systems with automatically clustered document collections and Web directories. Note that, the discussion is independent from the manner in which best-clusters are determined, which might be automatic or manual, and the clustering structure, i.e., partitioning or hierarchical. In the experiments, we use multiple query sets and the *Financial Times* database of TREC containing 210,158 documents of size 564 MB to evaluate the proposed algorithms. Our findings reveal that, selecting an appropriate integration algorithm based on the query size is crucial for reducing the query evaluation time.

The rest of the paper is organized as follows. In Section 2, we review the state of the art approaches for cluster searches (CS) using inverted index structures. In Section 3, we describe several query processing alternatives for result integration in cluster searches. In Section 4, the experimental environment is described and the efficiency figures of proposed strategies are extensively evaluated in Section 5. Finally, we conclude in Section 6.

## 2 State of the Art Approaches for Cluster Search (CS)

As mentioned before, there are two stages of information retrieval for clustered document collections [4, 5]: determining the best-matching clusters to a given query and then computing the best-matching documents from these clusters. During the best-document selection, a complementary step can also be required: integrating the best-cluster set information to the process of best-document selection to obtain the final query result. In Table 1, we list possible file structures to be used for best-document selection stage. We assume that a set of best clusters within which the search will be conducted is already obtained either automatically [4, 5] or manually, i.e., by browsing.

There are recent proposals that aim to eliminate the result integration step, either totally or partially, by using a modified document inverted index file during the best-document selection stage [1, 2, 3, 5]. In what follows, we first review the potential CS implementations that are based on typical document IIS and require a result integration step. After that, we discuss the implementations with modified inverted files that do not require any integration.

**Table 1.** Typical file structures for best-document selection stage

| **Best-document selection** | |
|---|---|
| **Initial best-document selection** | **Result integration** |
| Document Vectors (DV) | Cluster-document (CD) IIS |
| Document IIS (IIS) | Document-cluster (DC) IIS |

### 2.1 Cluster Search Implementations with the Result Integration Step

The best-document selection stage can use two file structures: DV (document vectors) and document IIS (inverted index of documents). In this paper, we assume that the documents are represented by vectors, based on vector space model [9]. The DV file includes the actual document vectors, whereas the IIS file stores an inverted index of all documents. An inverted index has a header part, including list of terms (vocabulary) encountered in the collection, and pointers to the posting lists for each term. A posting list for a term consists of the document ids that include the term and is usually ordered according to the document ids or some importance or relevance function that can optimize query processing. Note that, in all practical systems, a document IIS is used to compute query-document similarity, since the DV option is extremely costly (see [4] for a performance evaluation).

Once the best-clusters and best-documents are obtained separately, there are two ways to eliminate the best-documents that are not a member of the best-clusters [2, 3], i.e., to integrate the results of best-cluster and document selection stages. We call these alternatives "*document-id intersection based integration*" and "*cluster-id intersection based integration,*" and describe in detail next.

- *Document-id intersection based integration*: This alternative uses an inverted index such that for each cluster, the documents that fall into this particular cluster are stored (i.e., cluster-document (CD)-IIS). In this case, by using this latter (inverted) index, first the union of all documents that are within the best-clusters are determined, and then the resulting document set is intersected with the best-documents to obtain the final result. Note that, in an IR environment with clustering, such an inverted index of documents per cluster (i.e., a member document list for each cluster) is required in any case, to allow the browsing functionality.
- *Cluster-id intersection based integration*: The second integration alternative is just the reverse: for each document in the best-document set, the cluster(s) in which this document lies is found by using an (inverted) index that stores the list of clusters for each document (i.e., document-cluster (DC)-IIS). Then, the obtained cluster id(s) are intersected with the best-clusters set and if the result is not empty, the document is added to the final query output set.

The first integration alternative would be efficient when the number of documents per cluster is relatively small, whereas the second approach would be more efficient when the best-document set to be processed is small. Also note that, the inverted index required by the second alternative is redundant, as it is the transpose of the CD-IIS that would be implemented in any case to support the browsing functionality. On the other hand, as the integration process required by the first alternative requires first obtaining a union of several document lists and then an intersection, it would be less efficient in terms of query processing time, whereas storing an additional inverted index (DC-IIS) is not a major concern given the storage capabilities of modern systems [2]. In this paper, we focus on the algorithms for the second alternative, cluster-id intersection based integration, which seems to be more practical for large-scale IR systems.

### 2.2 Cluster Search Implementations Without the Result Integration Step

To avoid the integration step mentioned above, modified document inverted index files are proposed. In [2, 3], document identifiers are created as signatures, which convey information about the hierarchy of clusters in which a document belongs to. However, since the signatures can produce false drops, i.e. only provide an approximate filtering of best-document set, there is still a need for the cluster-id based integration approach to obtain the final query result. In [1, 5], we propose a skip-based IIS that differs from a typical IIS since in posting lists it stores the documents of each cluster in a group adjacent to each other. Remarkably, this skip-based approach is the only one that is fully evaluated for environments with compression [1] and doesn't require any integration step.

Notice that, our work presented here aims to discuss the efficiency of integration algorithms for systems using typical inverted index files. We believe that such large-scale IR systems exist due to possible data and application-specific limitations and reasons (e.g., a modified IIS may not be available if an off-the-shelf product is used to create the inverted index). Thus, present work would be valuable for those IR systems and Web directories employing typical inverted files.

## 3 Query Processing Algorithms for Cluster-Id Intersection Based Result Integration in Cluster Search

In this paper, we propose algorithms for the cluster-id based result integration approach for CS. It is assumed that the best-clusters set is already obtained automatically (i.e., by query-cluster centroid matching) or manually (i.e., by browsing). Then, a typical ranking query evaluation algorithm that can be employed during the best-document selection would be as shown in Figure 1.

The typical query evaluation as shown in Figure 1 works as follows. For each query term, corresponding posting list is retrieved, and for each document in the posting list, its accumulator array entry is updated by using a similarity measure (such as the cosine measure [10]). When all query terms are processed, a *min-heap* is used to extract the top scoring documents from the accumulator array. The details of this process can be found in [7, 10].

**Input:** Keyword-based query q
**Output:** Top-*k* best-matching documents
**In-memory data structures:** Document accumulator *DAcc*

(1) For each query term *t*
(2) Retrieve $I_t$, the posting list of term *t* from document IIS.
(3) For each <*doc-id, tf*> in $I_t$
(4) Update accumulator array entry *DAcc [doc-id]*
(5) Build a min-heap of size *k* for nonzero accumulator entries
(6) Extract the top-*k* best-matching documents from the heap

**Fig. 1.** Typical ranking query evaluation algorithm for keyword-based queries

To achieve within-cluster search, once the best-clusters are selected, the best-document selection phase can be implemented in different ways to integrate the best-cluster information. The alternatives differ in answering the following questions: (i) At what point during best-document selection should the cluster-id(s) of a particular document be intersected with the best-cluster ids, and (ii) What kind of data structure should be used to keep best-cluster ids? During query evaluation shown in Figure 1, the cluster ids can be intersected at three different points, yielding three implementation alternatives: (i) before updating the accumulator array for a document, (ii) before inserting a document to the min-heap, or (iii) after extracting the top scoring documents from the min-heap. Two potential data structures to store best-cluster ids are (i) a sorted array of best-clusters, or (ii) a 0/1 mark array in which entries for best clusters are 1 and all others are 0. We discuss these alternatives and their trade-offs in the following.

**Intersect Before Update (IBU).** In this approach (Figure 2), only those accumulator entries that belong to documents from best-clusters are updated. To achieve this, after a posting list is retrieved for a query term, the cluster to which each document in the posting list belongs is determined and intersected with the best-cluster set. If the document's cluster is found in the best-cluster set, its accumulator entry is updated.

Note that, this alternative would increase the efficiency of the last two steps of the algorithm (i.e., building and extracting from the heap as shown in lines 7-8), since all

**Input:** Keyword-based query q, best-clusters *BestClus*
**In-memory data structures:** Document accumulator *DAcc,* Document-category (DC) IIS

(1) For each query term *t*
(2) Retrieve $I_t$, the posting list of term *t* from document IIS.
(3) For each <*doc-id, tf*> in $I_t$
(4) Retrieve $I_{doc\text{-}id}$ from DC-IIS and obtain *Clus(doc-id)*
(5) If *Clus(doc-id)*$\cap$ *BestClus* $\neq \emptyset$
(6) Update accumulator array entry *DAcc [doc-id]*
(7) Build a min-heap of size *k* for nonzero accumulator entries
(8) Extract the top-*k* best-matching documents from the heap

**Fig. 2.** The query processing algorithm for intersect before insert (IBU) approach

the nonzero entries in the accumulator array are for the documents that are from best-clusters. On the other hand, the performance of this approach crucially depends on the cost of determining the clusters to which a document belongs (step 4) and cluster-id intersection operation (step 5). For the former operation, the algorithm should access document-cluster (DC) IIS for each element of the posting lists. However, if document-cluster associations are kept in the main memory or cached efficiently, this cost can be avoidable. This seems reasonable, since DC-IIS can be expected to be relatively small in size and can be shared among several query processing threads. For instance, assuming that documents are not repeated in more than one clusters, the main memory requirement to cache the entire DC-IIS would be only O($D$), i.e., in the order of document accumulator array. In this paper, without loss of generality, we assume that each document belongs to at most one cluster and the DC-IIS is stored in the main memory.

The cost of cluster ids' intersection is (assuming each document belongs to only one cluster) O(log $S$), if a sorted array of size $S$ is used to store best-cluster ids; and O(1) if a 0/1 mark array is used for this purpose. Note that, the data structure for best-clusters can be a sorted array if the memory reserved per query is scarce and/or total number of clusters is quite large. In this case, the document's cluster id can be searched within best-clusters using binary search. A 0/1 mark array is obviously more efficient but can only be preferred if the memory is not a concern and/or number of clusters is relatively small.

**Intersect Before Insert (IBI).** In this approach, instead of checking the cluster id intersection for each doc-id in each posting list, we do it once for each non-zero accumulator entry while building the heap (Figure 3). This alternative is preferable if the number of non-zero accumulator entries is expected to be low and/or the cost of cluster id intersection is high.

**Input:** Keyword-based query q, best-clusters *BestClus*
**In-memory data structures:** Document accumulator *DAcc,* Document-category (DC) IIS

(1) For each query term *t*
(2) Retrieve $I_t$, the posting list of term *t* from document IIS.
(3) Update accumulator array entry *DAcc [doc-id]*
(4) For each DAcc[doc-id] ≠ 0
(5) Retrieve $I_{doc\text{-}id}$ from DC-IIS and obtain *Clus(doc-id)*
(6) If *Clus(doc-id)*∩ *BestClus* ≠ Ø
(7) Insert into a min-heap of size *k*
(8) Extract the top-*k* best-matching documents from the heap

**Fig. 3.** The query processing algorithm for intersect before insert (IBI) approach

**Intersect After Extract (IAE).** In this third approach (Figure 4) the entire query processing works as in Figure 1 and only at the end of the evaluation, the cluster-ids of top-*k* documents are intersected with the best-clusters. Of course, if some of those *k* documents are not from the best-clusters, then the build-heap step and extraction should be repeated. To avoid such a repetition, the initial evaluation can be executed for top-$L$ documents, $L > k$. In this case, the cost of cluster-id intersection is

negligible as it is postponed at the end of processing and $L << D$. On the other hand, it is important to choose $L$ appropriately, if $L$ is much larger than $k$ (e.g., $L = D$ the extreme case), the gains in the intersection stage would be lost during the build-heap and extraction. If $L$ is too small (i.e., very close to $k$), we may need more than one iteration to find $k$ documents that are in the best-clusters. Thus, IAE alternative will be useful if it can somehow be guaranteed that in a small number of highest scoring documents, there will be at least $k$ documents from the best clusters. More specifically, this approach would be better than the previous alternative only if cluster intersection is costly; and better than the IBU algorithm if both intersection test is expensive and too many nonzero accumulator entries arise.

**Input:** Keyword-based query q, best-clusters *BestClus*
**Output:** Top-*k* best-matching documents *Result*
**In-memory data structures:** Document accumulator *DAcc,* Document-category (DC) IIS

(1) For each query term *t*
(2) Retrieve $I_t$, the posting list of term *t* from document IIS.
(3) For each <*doc-id, tf*> in $I_t$
(4) Update accumulator array entry *DAcc [doc-id]*
(5) Build a min-heap of size *L* (*L>k)* for nonzero accumulator entries
(6) Extract the top-*L* best-matching documents from the heap
(7) While size of *Result* < k
(8) For each doc-id ∈ top-*L*
(9) Retrieve $I_{doc\text{-}id}$ from DC-IIS and obtain *Clus(doc-id)*
(10) If *Clus(doc-id)*∩ *BestClus* ≠ Ø
(11) Insert doc-id to the *Result*
(12) If size of *Result* < *k*
(13) Set *L* to *M* for some *M* > *L*, go to step (5)

**Fig. 4.** The query processing algorithm for intersect after extract (IAE) approach

## 4 Experimental Environment

**Document Database and Clustering Structure.** In the experiments, *Financial Times* document collection (referred to as the FT database) of TREC Disk 4 is used. The document database includes *210,158* newspaper articles published between 1991 and 1994. The indexing process with the elimination of English stop-words and numbers yields a lexicon of 229,748 terms.

The database is clustered using C3M algorithm [6] in partitioning mode, which yields 1640 clusters and 128 documents per clusters, on the average. An important parameter is the number of best-matching clusters, and following the common practice in earlier works [5] we use 10% of the total number of clusters (i.e., 164 clusters) as the number of best-clusters in the retrieval experiments. The clustering structure and other parameters are validated for FT database in our previous study [5]. In this paper, we provide results for retrieving top-10 documents, i.e., $k$ =10.

**Queries, Query Matching & Centroid Weighting.** We used the TREC-7 query topics corresponding to the FT database of TREC Disk 4 collection (queries 351-400).

In the experiments we use three different types of query sets: *Qshort*, *Qmedium*, *Qlong* including 2.38, 8.16 and 190 terms on the average, respectively. The first two of the query sets are created from the TREC queries, namely *Qshort* queries include TREC query titles, and *Qmedium* queries include both titles and descriptions. The third one, *Qlong*, is created from the top retrieved document of each *Qmedium* query. The *Qlong* set, with extremely long queries, represents "find similar documents" type queries supported by typical Web search engines.

In this study, the document term weights are assigned using the *term frequency* x *inverse document frequency* (IDF) information and using a well-known term weighting formula. During query processing, term weights are normalized by using the document lengths. The term weights for query terms are calculated using the augmented normalized frequency formula [5]. After obtaining weighted document ($d$) and query ($q$) vectors in an $n$ dimensional vector space the query-document matching is performed using the well-known cosine formula [5, 10].

For the cluster centroids, we take a simplistic approach and use all cluster member documents' terms as centroid terms. The weight of a centroid term is also computed by the formula *term frequency* x *IDF*. Please see [5] for further details.

## 5 Experimental Results

The experiments are conducted on a Pentium 4 2.54 GHz PC with 2GB memory and Mandrake Linux operating system. All implementations use C programming language. Unless otherwise stated, 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 available in memory simultaneously.

Along with the lines of Section 3, we discuss three query processing implementations (IBU, IBI, IAE) and two versions for each such implementation –the version that uses a sorted array (SA) to keep and look up best-clusters, and the version that uses a 0/1 mark array (MA) for the same purpose. During query evaluation, first the queries in the test sets are matched with the cluster centroids to obtain the best-matching clusters (top 10% of clusters). Next, best-documents within these best-clusters are computed using the three possible algorithms with two different data structures (SA, MA) for best clusters. In Table 2, we report in-memory processing time during best-document selection for each strategy, as well as the average number of accumulator update operations, number of nonzero document accumulator entries, number of cluster-id intersection operations and finally number of heap insertion operations.

From Table 2, the following observations can be drawn. For the short and medium length queries (i.e., as in the cases of *Qshort* and *Qmedium* sets) IAE approach is inferior to other two algorithms due to very high costs of build-heap and extract operations. As shown in Table 2, number of heap insertion operations is at least 10 times larger with respect to other algorithms. Note that, in these experiments, we choose $L$ (i.e., the min-heap size) as the total number of documents in the entire collection, which is the extreme condition, to avoid repeating the heap build and extraction steps as discussed in Section 3. If $L$ is set to $k$ (=10), the efficiency of this

**Table 2.** Efficiency comparison of the integration algorithms (IBU: Intersect Before Update, IBI: Intersect Before Insert, IAE: Intersect After Extract, SA: Sorted Array, MA: Mark Array)

| | Time (sec) and operation counts (all averages) | IBU-SA | IBU-MA | IBI-SA | IBI-MA | IAE (SA & MA) |
|---|---|---|---|---|---|---|
| ***Qshort*** | Query evaluation time | 0.007 | **0.005** | 0.007 | 0.006 | 0.012 |
| | No. of accumulator updates | 908 | 908 | 9792 | 9792 | 9792 |
| | No. of nonzero accumulators | 848 | 848 | 9462 | 9462 | 9462 |
| | No. of intersections | 9792 | 9792 | 9462 | 9462 | 65 |
| | No. of heap insertion calls | 848 | 848 | 848 | 848 | 9462 |
| ***Qmedium*** | Query evaluation time | 0.018 | **0.008** | 0.017 | 0.010 | 0.044 |
| | No. of accumulator updates | 3786 | 3786 | 49416 | 49416 | 49416 |
| | No. of nonzero accumulators | 2899 | 2899 | 39496 | 39496 | 39496 |
| | No. of intersections | 49416 | 49416 | 39496 | 39496 | 51 |
| | No. of heap insertion calls | 2899 | 2899 | 2899 | 2899 | 39496 |
| ***Qlong*** | Query evaluation time | 0.448 | 0.111 | 0.133 | **0.102** | 0.338 |
| | No. of accumulator updates | 124115 | 124115 | 1.8 mil. | 1.8 mil. | 1.8 mil. |
| | No. of nonzero accumulators | 11718 | 11718 | 189510 | 189510 | 189510 |
| | No. of intersections | 1.8 mil. | 1.8 mil. | 189510 | 189510 | 27 |
| | No. of heap insertion calls | 11718 | 11718 | 11718 | 11718 | 189510 |

approach also improves significantly (i.e. 0.006, 0.010 and 0.107 versus 0.012, 0.044 and 0.338 seconds for *Qshort*, *Qmedium* and *Qlong*, respectively); however there is always the possibility that all of these top-k documents are not from best clusters; which would require building and extracting from a larger min-heap. Also note that, since IAE-SA and IAE-MA approaches do not differ significantly in terms of performance, their efficiency figures are shown in the same column in Table 2.

On the other hand, assuming that DC-IIS is kept in the main memory, the performance of IBU-SA and IBI-SA approaches seem to be very similar, the same is true for the IBU-MA and IBI-MA approaches. Clearly, the versions that employ a 0/1 mark array to store best clusters are faster than their sorted array based counterparts. If the memory is large enough to keep DC-IIS in memory, IBU-MA approach performs better than IBI-MA and provides up to 15% and 20% reductions in query processing times for *QShort* and *Qmedium*, respectively. If it is impossible to keep DC-IIS in memory, the IAE method with the minimum number of cluster-id intersection operations would be the method of choice, however we envision that this case may not be highly probable given the modern systems' memory capacities. For instance, in our experimental setup, the size of DC-IIS is only around 1 MB.

For very long queries (as in the case of *Qlong* set), again IBU and IBI approaches with MA seem to be the most reasonable implementation candidates given that DC-IIS in in-memory. For this case, IBU-SA suffers from the excessive cost of cluster-id intersection operations and performs even worse than IAE; so if IBU is the choice of implementation, it should be coupled with MA data structure. Nevertheless, IBI-MA approach outperforms IBU-MA in an 8% margin and seems to be the most efficient approach. As before, IAE (with SA or MA) may only be chosen if DC-IIS can not be stored or cached in the main memory.

Our findings show that, depending on the query set properties and main memory availability to store the DC-IIS and best-clusters, the most appropriate query processing approach for within-cluster search should be determined dynamically by the IR system.

## 6 Conclusion

In this paper, we propose and evaluate within-cluster search algorithms to efficiently integrate the best-clusters and best-documents for cluster-based IR systems using inverted index structures. Our findings reveal that the efficiency of the integration algorithm depends on the query length, and the appropriate algorithm should be chosen dynamically by the IR system considering query properties and available system resources.

**Acknowledgments.** This work is partially supported by The Scientific and Technical Research Council of Turkey (TÜBİTAK) under the grant no 105E024.

## References

1. Altingovde, I.S., Can, F., Demir, E., Ulusoy, O.: Incremental cluster-based retrieval with embedded centroids using compressed cluster-skipping inverted files. Submitted for publication.
2. Cacheda, F., Baeza-Yates, R.: An optimistic model for searching Web directories. In: Proceedings of the 26th European Conf. on IR Research, Sunderland, UK. (2004) 364–377
3. Cacheda, F., Carneiro, V., Guerrero, C., Viña, Á.: Optimization of restricted searches in Web directories using hybrid data structures. In: Proceedings of the 25th European Conference on IR Research (ECIR), Pisa, Italy. (2003) 436–451
4. Can, F.: On the efficiency of best-match cluster searches. Information Processing and Management **30** (3) (1994) 343–361
5. Can, F., Altingovde, I.S., Demir, E.: Efficiency and effectiveness of query processing in cluster-based retrieval. Information Systems **29** (8) (2004) 697–717
6. Can, F., Ozkarahan E. A.: Concepts and effectiveness of the cover-coefficient-based clustering methodology for text databases. ACM TODS **15** (4) (1990) 483–517
7. Cambazoglu, B.B., Aykanat, C.: Performance of query processing implementations in ranking-based text retrieval systems using inverted indices. Information Processing and Management **42** (4) (2006) 875–898
8. van Rijsbergen, C. J.: Information retrieval. 2nd ed. Butterworths, London (1979)
9. Salton, G.: Automatic text processing: the transformation, analysis, and retrieval of information by computer. Addison Wesley, Reading, MA (1989)
10. Witten, I. H., Moffat, A., Bell, T. C.: Managing gigabytes compressing and indexing documents and images. Van Nostrand Reinhold, New York (1994)