XML Retrieval Using Pruned Element-Index Files

Ismail Sengor Altingovde, Duygu Atilgan, and Özgür Ulusoy

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

Abstract. An element-index is a crucial mechanism for supporting content-only (CO) queries over XML collections. A full element-index that indexes each element along with the content of its descendants involves a high redundancy and reduces query processing efficiency. A direct index, on the other hand, only indexes the content that is directly under each element and disregards the descendants. This results in a smaller index, but possibly in return to some reduction in system effectiveness. In this paper, we propose using static index pruning techniques for obtaining more compact index files that can still result in comparable retrieval performance to that of a full index. We also compare the retrieval performance of these pruning based approaches to some other strategies that make use of a direct element-index. Our experiments conducted along with the lines of INEX evaluation framework reveal that pruned index files yield comparable to or even better retrieval performance than the full index and direct index, for several tasks in the ad hoc track.

1 Introduction

Classical information retrieval (IR) is the quest for identifying the documents in a collection that are most relevant to a user's information need, usually expressed as a keyword query. Although there have been efforts for passage retrieval, most of the previous works in the IR field presume a document as the typical unit of retrieval. In contrary, XML documents, which started to emerge since late 90s, have a logical structure and may allow finer-grain retrieval, i.e., at the level of elements. Given that XML is used for the representation of lengthy documents, such as e-books, manuals, legal transcripts, etc., the "focused" retrieval is expected to provide further gains for the end users in locating the specific relevant information [16].

In the last decade, especially under the INitiative for the Evaluation of XML retrieval (INEX) [13] campaigns, several indexing, ranking and presentation strategies for XML collections have been proposed and evaluated. Given the freshness of this area, there exists a number of issues that are still under debate. One such fundamental problem is indexing the XML documents. As mentioned above, the focused retrieval aims to identify the most relevant parts of an XML document to a query, rather than retrieving the entire document. This requires constructing an index at a lower granularity, say, at the level of elements, which is not a trivial issue given the nested structure of XML documents.

In this paper, we essentially focus on the strategies for constructing space-efficient element-index files to support content-only (CO) queries. In the literature, the most

C. Gurrin et al. (Eds.): ECIR 2010, LNCS 5993, pp. 306–318, 2010.

straight-forward element-indexing method considers each XML element as a separate document, which is formed of the text directly contained in it and the textual content of all of its descendants. We call this structure a *full* element-index. Clearly, this approach yields significant redundancy in terms of the index size, as elements in the XML documents are highly nested. To remedy this, a number of approaches, as reviewed in detail in the next section, are proposed in the literature. One such method is restricting the set of elements indexed, based on the size or type of the elements [18]. Such an approach may still involve redundancy for the elements that are selected to be indexed. Furthermore, determining what elements to index would be collection and scenario dependent. There are also other indexing strategies that can fully eliminate the redundancy. For instance, a *direct* element-index is constructed by only considering the text that is directly contained under an element (i.e., disregarding the content of the element's descendents). In the literature, propagation based mechanisms, in which the score of each element is propagated upwards in the XML structure, are coupled with the direct index for effective retrieval (e.g., [11]). In this case, the redundancy in the index is somewhat minimized, but query processing efficiency would be degraded.

For an IR system, it is crucial to optimize the underlying inverted index size for the efficiency purposes. A lossless method for reducing the index size is using compression methods (see [22] for an exhaustive survey). On the other hand, many lossy static index pruning methods are also proposed in the last decade. All of these methods aim to reduce the storage space for the index and, subsequently, query execution time, while keeping the quality of the search results unaffected. While it is straight-forward to apply index compression methods to (most of) the indexing methods proposed for XML, it is still unexplored how those pruning techniques serve for XML collections, and how they compare to the XML-specific indexing methods proposed in the literature.

In this paper, we propose to employ static index pruning techniques for XML indexing. We envision that these techniques may serve as a compromise between a full element-index and a direct element-index. In particular, we first model each element as the concatenation of the textual content in its descendants, as typical in a full index. Then, the redundancy in the index is eliminated by pruning this initial index. In this way, an element is allowed to contain some terms, say, the most important ones, belonging to its descendants; and this decision is given based on the full content of the element in an adaptive manner.

For the purposes of index pruning, we apply two major methods from the IR literature, namely, term-centric [5] and document-centric pruning [4] to prune the full element-index. We evaluate the performance for various retrieval tasks as described in the latest INEX campaigns. More specifically, we show that retrieval using pruned index files is comparable or even superior to that of the full index up to very high levels of pruning. Furthermore, we compare these pruning-based approaches to a retrieval strategy coupled with a direct index (as in [11]) and show that pruning-based approaches are also superior to that strategy. As another advantage, the pruning-based approaches are more flexible and can reduce an index to a required size.

In the next section, we first review a number of indexing strategies for XML collections as well as the associated retrieval techniques to support content-only queries. Next, we summarize some of the static index pruning strategies that are proposed for large-scale IR systems and search engines. In Section 3, we describe the pruning

techniques that are adapted for reducing the size of a full element-index. Section 4 is devoted to the experimental evaluations. Finally, we conclude and point to future work directions in Section 5.

2 Related Work

2.1 Indexing Techniques for XML Retrieval

In the literature, several techniques are proposed for indexing the XML collections and for query processing on top of these indexes. In a recent study, Lalmas [16] provides an exhaustive survey of indexing techniques—essentially from the perspective of IR discipline—that we briefly summarize in the rest of this section.

The most straight-forward approach for XML indexing is creating a "full" element-index, in which each element is considered along with the content of its descendants. In this case, how to compute inverse document frequency (IDF), a major component used in many similarity metrics, is an open question. Basically, IDF can be computed across all elements, which also happens to be the approach taken in our work. As a more crucial problem [16], a full element-index is highly redundant because the terms are repeated for each nested element and the number of elements is typically far larger than the number of documents.

To cope with the latter problem, an indexing strategy can only consider the direct textual content of each element, so that redundancy due to nesting of the elements would be totally removed. In [9, 10], only leaf nodes are indexed, and the scores of the leaf elements are propagated upwards to contribute to the scores of the interior (ancestor) elements. In a follow-up work [11], the direct content of each element (either leaf or interior) is indexed, and again a similar propagation mechanism is employed. Another alternative is propagating the representations of elements, e.g., term statistics, instead of the scores. However, the propagation stage, which has to be executed during the query processing time, can degrade the overall system efficiency.

In the database field, where XML is essentially considered from a data-centric rather than a document-centric point of view, a number of labeling schemes are proposed especially to support structural queries (see [20] for a survey). In XRANK system [12], postings are again created only for the textual content directly under an element; however document identifiers are encoded using the Dewey ids so that the scores for the ancestor elements can also be computed without a propagation mechanism. This indexing strategy allows computing the same scores as a full index while the size of the index can be in the order of a direct index. However, this scheme may suffer from other problems, such as the excessive Dewey id length for very deeply located elements. We provide a further discussion of this strategy in Section 4.

An in-between approach to remedy the redundancy in a full element-index is indexing only certain elements of the documents in the collection. Element selection can be based upon several heuristics (see [16] for details). For instance, shorter elements (i.e., with only few terms) can be discarded. Another possibility is selecting elements based on their popularity of being assessed as relevant in INEX framework. The semantics of the elements can also be considered while deciding which elements to index by a system designer. Yet another indexing technique that is also related is

distributed indexing, which proposes to create separate indexes for each element type, possibly selected with one of the heuristics discussed above. This latter technique may be especially useful for computing the term statistics in a specific manner to each element type.

In this paper, our main concern is reducing the index size to essentially support content-only queries. Thus, we attempt to make an estimation of how the index sizes for the above approaches can be ordered. Of course, I_{full}, i.e., full element-index, would have the largest size. Selective (and/or distributed) index, denoted as I_{sel}, would possibly be smaller; but it can still involve some degree of redundancy for those elements that are selected for indexing. Thus, a direct index (I_{direct}) that indexes only the text under each element would be smaller than the former. Finally, the lower bound for the index size can be obtained by discarding all the structuring in an XML document and creating an index only on the document basis (i.e., I_{doc}). Thus, a rough ordering can be like $size(I_{full}) \geq size(I_{sel}) \geq size(I_{direct}) \geq size(I_{doc})$. In this paper, we employ some pruning methods that can yield indexes of sizes comparable to $size(I_{direct})$ or $size(I_{sel})$. We envision that such methods can prune an index up to a given level in a robust and adaptive way, without requiring a priori knowledge on the collection (e.g., semantics or popularity of elements). Furthermore, the redundancy that remains in the index can even help improving the retrieval performance. These index pruning methods are reviewed in the next section.

2.2 Static Pruning Strategies for Inverted Indexes

In the last decade, a number of different approaches are proposed for the static index pruning. The underlying goal of static index pruning is to reduce the file size and query processing time, while keeping the search result quality and subsequently, the effectiveness of the system unaffected, or only slightly affected. In this paper, as in [4], we use the expressions *term-centric* and *document-centric* to indicate whether the pruning process iterates over the terms (or, equivalently, the posting lists) or the documents at the first place, respectively.

In one of the earliest works in this field, Carmel et al. proposed term-centric approaches with uniform and adaptive versions [5]. Roughly, adaptive top-k algorithm sorts the posting list of each term according to some scoring function (Smart's TF-IDF in [5]) and removes those postings that have scores under a threshold determined for that particular term. The algorithm is reported to provide substantial pruning of the index and exhibit excellent performance at keeping the top-ranked results intact in comparison to the original index. In our study, this algorithm (which is referred to as TCP strategy hereafter) is employed for pruning full element-index files, and it is further discussed in Section 3.

As an alternative to term-centric pruning, Büttcher et al. proposed a document-centric pruning (referred to as DCP hereafter) approach with uniform and adaptive versions [4]. In the DCP approach, only the most important terms are left in a document, and the rest are discarded. The importance of a term for a document is determined by its contribution to the document's Kullback-Leibler divergence (KLD) from the entire collection. In a more recent study [1], a comparison of TCP and DCP for pruning the entire index is provided in a uniform framework. It is reported that for disjunctive query processing TCP essentially outperforms DCP for various parameter

selections. In this paper, we also use the DCP strategy to prune the full element-index, and further discuss DCP in Section 3.

There are several other proposals for static index pruning in the literature. A locality based approach is proposed in [6] for the purposes of supporting conjunctive and phrase queries. In a number of other works, search engine query logs are exploited to guide the static index pruning [2, 8, 17, 19].

3 Pruning the Element-Index for XML Retrieval

The size of the full element-index for an XML collection may be prohibitively large due to the nested structure of the documents. A large index file does not only consume storage space but also degrades the performance of the actual query processing (as longer posting lists should be traversed). The large index size would also undermine the other optimization mechanisms, such as the list caching (as longer list should be stored in the main memory).

Previous techniques in the literature attempt to reduce the size of a full element-index by either totally discarding the overlapping content, or only indexing a subset of the elements in a collection. In contrast, we envision that some of the terms that appear in an element's descendants may be crucial for the retrieval performance and should be repeated at the upper levels; whereas some other terms can be safely discarded. Thus, instead of a crude mechanism, for each element, the decision for indexing the terms from the element's descendants should be given adaptively, considering the element's textual content and search system's ranking function. To this end, we employ two major static index pruning techniques, namely term-centric pruning (TCP) [5] and document-centric pruning (DCP) [4] for indexing the XML collections. Below, we outline these strategies as used in our study.

- *TCP(I, k, ε)*: As it is mentioned in the previous section, TCP, the adaptive version of the top-*k* algorithm proposed in [5], is reported to be very successful in static pruning. In this strategy, for each term t in the index I, first the postings in t's posting list are sorted by a scoring function (e.g, TF-IDF). Next, the k^{th} highest score, z_t, is determined and all postings that have scores less than $z_t * \varepsilon$ are removed, where ε is a user defined parameter to govern the pruning level. Following the practice in [3], we disregard any theoretical guarantees and determine ε values according to the desired pruning level.

  A recent study shows that the performance of the TCP strategy can be further boosted by carefully selecting and tuning the scoring function used in the pruning stage [3]. Following the recommendations of that work, we employ BM25 as the scoring function for TCP.
- *DCP(D, λ)*: In this paper, we apply the DCP strategy for the entire index, which is slightly different from pruning only the most frequent terms as originally proposed by [4]. For each document d in the collection D, its terms are sorted by the scoring function. Next, the top $|d|*\lambda$ terms are kept in the document and the rest are discarded, where λ specifies the pruning level. Then, the inverted index is created over these pruned documents.

KLD has been employed as the scoring function in [4]. However, in a more recent work [1], it is reported that BM25 performs better when it is used during both pruning and retrieval. Thus, we also use BM25 with DCP algorithm.

In this paper, we compare the retrieval performance of four different XML indexing approaches:

- I_{full}: Full element-index (as described before)
- I_{direct}: An index created by using only the text directly under each element
- $I_{TCP, \varepsilon}$: Index files created from I_{full} by using TCP algorithm at a pruning level ε
- $I_{DCP, \lambda}$: Index files created from I_{full} by using DCP algorithm at a pruning level λ.

4 Experiments

4.1 Experimental Setup

Collection and queries. In this paper, we use English Wikipedia XML collection [7] employed in INEX campaigns between 2006 and 2008. The dataset includes 659,388 articles obtained from Wikipedia. After conversion to XML, the collection includes 52 million elements. The textual content is 1.6 GB whereas the entire collection (i.e., with element tags) takes 4.5 GB.

Our main focus in this paper is content-only (CO) queries whereas content-and-structure queries (CAS) are left as a future work. In the majority of the experiments reported below, we use 70 query topics with relevance assessments provided for the Wikipedia collection in INEX 2008 (see [15; p. 8] for the exact list of the queries). The actual query set is obtained from the title field of these topics after eliminating the negated terms and stopwords. No stemming is applied.

Indexing. As we essentially focus on CO queries, the index files are built upon only using the textual content of the documents in the collection; i.e., tag names and/or paths are not indexed. In the best performing system in all three tasks of INEX 2008 ad hoc retrieval track, only a subset of elements in the collection are used for scoring [14]. Following the same practice, we only index the following elements: <p>, <section>, <normallist>, <article>, <body>, <td>, <numberlist>, <tr>, <table>, <definitionlist>, <th>, <blockquote>, , <li>, <u>. Each of these elements in an XML document is treated as a separate document and assigned a unique global identifier. Thus, the number of elements to be indexed is found to be 7.4 million out of 52 million elements in Wikipedia collection.

During indexing, we use the open-source Zettair search engine [21] to parse the documents in the collection and obtain a list of terms per element. Then, an element-level index is constructed by using each of the strategies described in this paper. The posting lists in the resulting index files include <element-id, frequency> pairs for each term in the collection, as this is adequate to support the CO queries. Of course, the index can be extended to include, say, term positions, if the system is asked to support phrase or proximity queries, as well. Posting lists are typically stored in a binary file format where each posting takes 8 bytes (i.e., a 4 byte integer is used per each field).

The resulting element-index takes 4 GB disk space. In the below discussions, all index sizes are considered in terms of their raw (uncompressed) sizes.

Retrieval tasks and evaluation. In this study, we concentrate on three ad-hoc retrieval tasks, namely, Focused, Relevant-in-Context (RiC) and Best-in-Context (BiC), as described in recent INEX campaigns (e.g., see [13, 15]). In short, the Focused task is designed to retrieve the most focused results for a query without returning overlapping elements. The underlying motivation for this task is retrieving the relevant information at the correct granularity. Relevant-in-Context task requires returning a ranked list of documents and a set of relevant (non-overlapping) elements listed for each article. Finally, Best-in-Context task is designed to find the best-entry-point (BEP) for starting to read the relevant articles. Thus, the retrieval system should return a ranked list of documents along with a BEP for each document.

We evaluate the performance of different XML indexing strategies for all these three tasks along with the lines of INEX 2008 framework. That is, we use INEXeval software provided in [13] which computes a number of measures for each task, which is essentially based on the amount of retrieved text that overlaps with the relevant text in assessments. In all experiments, we return up to 1500 highest scoring results.

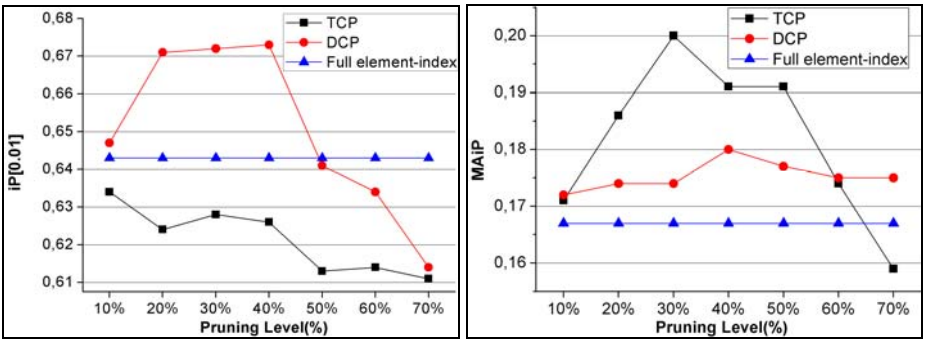
4.2 Performance Comparison of Indexing Strategies: Focused Task

For the focused retrieval task, we return the highest scoring 1500 elements after eliminating the overlaps. The overlap elimination is simply achieved by choosing the highest scoring element on a path in the XML document.

In Figure 1a, we plot the performance of TCP and DCP based indexing strategies with respect to the full element-index, I_{full}. The evaluation measure is interpolated precision at 1% recall level, iP[0.01], which happens to be the official measure of INEX 2008. For this experiment, we use BM25 function as described in [4] to rank the elements using each of the index files. For the pruned index files, the element length, i.e., number of terms in an element, reduces after pruning. In earlier studies [1, 3], it is reported that using the updated element lengths results better in terms of effectiveness. We observed the same situation also for XML retrieval case, and thus, use the updated element lengths for each pruning level of TCP and DCP.

To start with, we emphasize that the system performance with I_{full} is reasonable in comparison to INEX 2008 results. That is, focused retrieval based on I_{full} yields an iP figure of 0.643 at 1% recall level. The best official result in INEX 2008 for this task is 0.689 and our result is within the top-10 results of this task (see Table 6 in [15]). This is also the case for RiC and BiC results that will be discussed in the upcoming sections, proving that we have a reasonable baseline for drawing conclusions in our experimental framework.

Figure 1a reveals that DCP based indexing is as effective as I_{full} up to 50% pruning and indeed, at some pruning levels, it can even outperform I_{full}. In other words, it is possible to halve the index and still obtain the same or even better effectiveness than the full index. TCP is also comparable to I_{full} up to 40%. For this setup, DCP seems to be better than TCP, an interesting finding given that just the reverse is observed for typical document retrieval in previous works [1, 2]. However, the situation changes for higher levels of recall (as shown in Table 1), and, say, for iP[.10], TCP performs better than DCP up to 70% pruning.



(a) **(b)**

Fig. 1. Effectiveness comparison of I_{full}, I_{TCP} and I_{DCP} in terms of (a) iP[0.01], and (b) MAiP

In Table 1, we report the interpolated precision at different recall levels and mean average interpolated precision (MAiP) computed for 101 recall levels (from 0.00 to 1.00). For these experiments, due to lack of space, we only show three pruning levels (30%, 50% and 70%) for both TCP and DCP. The results reveal that, up to 70% pruning, both indexing approaches lead higher MAiP figures than I_{full} (also see Figure 1b). The same trend also applies for iP at higher recall levels, namely, 5% and 10%.

In Table 1, we further compare the pruning based approaches to other retrieval strategies using I_{direct}. Recall that, as discussed in Section 3, I_{direct} is constructed by considering only the textual content immediately under each element, disregarding the element's descendants. For this collection, the size of the index turns out to be almost 35% of the I_{full}, i.e., corresponding to 65% pruning level. In our first experiment, we evaluate focused retrieval using I_{direct} and BM25 as in the above. In this case, I_{direct} also performs well and yields 0.611 for iP[.01] measure, almost the same effectiveness for slightly smaller indexes created by TCP and DCP (see the case for 70% pruning for TCP and DCP in Table 1). However, in terms of iP at higher levels and MAiP, I_{direct} is clearly inferior to the pruning based approaches, as shown in Table 1.

As another experiment, we decided to implement the propagation mechanism used in the GPX system that participated in INEX between 2004 and 2006 [9, 10, 11]. In these campaigns, GPX is shown to yield very competitive results and ranked among the top systems for various retrieval tasks. Furthermore, GPX is designed to work with an index as I_{direct}, i.e., without indexing the content of descendants for the elements. In this system, first, the score of every element is computed as in the typical case. However, before obtaining the final query output, the scores of all elements in an XML document are propagated upwards so that they can also contribute to their ancestors' scores.

In this paper, we implemented this propagation mechanism (denoted as PROP) of GPX and used Equation 2 given in [11]. In accordance with the INEX official run setup described in that work, we set the parameters *N1*=0.11 and *N2*=0.31 in our implementation. Their work also reports that another scoring function (denoted as SCORE here) performs quite well (see Equation 1 in [11]) when coupled with the propagation. Thus, we obtained results for the propagation mechanism using both scoring functions,

Table 1. Effectiveness comparison of indexing strategies for Focused task. Prune (%) field denotes the percentage of pruning with respect to full element-index (I_{full}). Shaded measures are official evaluation measure of INEX 2008. Best results for each measure are shown in bold.

Indexing Strategy	Prune (%)	iP[.00]	iP[.01]	iP[.05]	iP[.10]	MAiP
I_{full}	0%	0.725	0.643	0.507	0.446	0.167
$I_{TCP, 0.3}$	30%	0.700	0.628	**0.560**	0.511	**0.200**
$I_{DCP, 0.3}$	30%	**0.750**	**0.672**	0.529	0.469	0.174
$I_{TCP, 0.5}$	50%	0.666	0.613	0.549	**0.518**	0.191
$I_{DCP, 0.5}$	50%	0.708	0.641	0.518	0.473	0.177
$I_{TCP, 0.7}$	70%	0.680	0.611	0.511	0.446	0.159
$I_{DCP, 0.7}$	70%	0.681	0.614	0.534	0.477	0.175
I_{direct}	65%	0.731	0.611	0.448	0.362	0.126
I_{direct}+$PROP_{SCORE}$	65%	0.519	0.473	0.341	0.302	0.110
I_{direct}+$PROP_{BM25}$	65%	0.450	0.435	0.384	0.302	0.116

namely, BM25 and SCORE. In Table 1, corresponding experiments are denoted as I_{direct}+$PROP_{BM25}$ and I_{direct}+$PROP_{SCORE}$, respectively. The results reveal that, SCORE function performs better at early recall levels, but for both cases the effectiveness figures are considerably lower than the corresponding results (i.e., 70% pruning level) based on TCP and DCP. We attribute the lower performance of PROP mechanism to the following observation. For the Wikipedia dataset, 78% of the data assessed as relevant resides in the leaf nodes. This means that, returning leafs in the result set would improve effectiveness, and vice versa. In contrast, PROP propagates element scores to the upper levels in the document, which may increase the number of interior nodes in the final result and thus reduce the effectiveness.

Note that, we also attempted to verify the reliability of our implementation of propagation mechanism by using INEX 2006 topics and evaluation software, and to see how our results compare to the GPX results reported in [11]. We observed that the results slightly differ at early ranks but then match for higher rank percentages.

We can summarize our findings as follows: In terms of the official INEX measure, which considers the performance at the first results most, the index files constructed by the static pruning techniques lead to comparable to or even superior results than I_{full} up to 70% pruning level. A direct element-index also takes almost 35% of the full index. Its performance is as good as the pruned index files for iP[.01], but it falls rapidly at higher recall levels. Score propagating retrieval systems, similar to GPX, perform even worse with the I_{direct} and do not seem to be a strong competitor.

Finally, there is another indexing approach proposed in the XRANK system [12] that can serve as a natural competitor of the strategies discussed here. In the XRANK's approach, as reviewed in Section 2, element ids are represented with Dewey encoding. This representation can yield an index of size I_{direct}, while the exact element scores that could be obtained from a full element-index can also be computed (with some overhead during query processing). Furthermore, it can support structure based constraints for CAS queries. However, this indexing technique may also cause some problems. For instance, since element ids are Dewey encoded, it may be hard to represent some elements in deeply nested XML documents. Another issue may be updating the Dewey codes when an XML document is updated (see [20] for a general discussion). Also, to our best knowledge, the performance of typical inverted index

Table 2. Effectiveness comparison of indexing strategies for RiC task

Indexing Strategy	Prune (%)	gP[5]	gP[10]	gP[25]	gP[50]	MAgP
I_{full}	0%	0.364	**0.321**	0.246	0.198	0.190
$I_{TCP, 0.3}$	30%	0.380	0.321	0.248	0.199	0.193
$I_{DCP, 0.3}$	30%	0.381	0.321	**0.256**	**0.202**	**0.196**
$I_{TCP, 0.5}$	50%	**0.385**	0.321	0.247	0.196	0.185
$I_{DCP, 0.5}$	50%	0.366	0.321	0.252	0.196	0.185
$I_{TCP, 0.7}$	70%	0.355	0.297	0.223	0.170	0.152
$I_{DCP, 0.7}$	70%	0.352	0.305	0.232	0.172	0.157
I_{direct}	65%	0.312	0.281	0.210	0.168	0.140
I_{direct}+$PROP_{SCORE}$	65%	0.275	0.242	0.201	0.158	0.142
I_{direct}+$PROP_{BM25}$	65%	0.223	0.199	0.184	0.145	0.108

compression techniques for Dewey encoded index files is not evaluated yet. On the other hand, the index files created by the static pruning techniques can be processed by typical IR systems without requiring any modifications in the query processing, indexing and compression modules. Additionally, as shown in above results, these techniques may yield better effectiveness than a full index in several cases. Nevertheless, we leave comparison with the XRANK's approach as a future work.

4.3 Performance Comparison of Indexing Strategies: Relevant-in-Context Task

For the Relevant-in-Context task, after scoring the elements, we again eliminate the overlaps and determine the top-1500 results as in the Focused task. Then, those elements from the same document are grouped together. The result is a ranked list of documents along with a set of elements. While ranking the documents, we use the score of the highest scoring retrieved element per document as the score of this particular document. We also experimented with another technique in the literature; i.e. using the average score of elements in a document for ranking the documents, which performs worse than the former approach and is not investigated further.

In Table 2, we present the results in terms of the generalized precision (gP) metric at early ranks and mean average generalized precision (MAgP), i.e., the official measure of INEX 2008 for both RiC and BiC tasks. The results show that approaches using I_{direct} are inferior to those using the pruned index files based on either TCP or DCP at 70% pruning level; however with a relatively smaller margin with respect to the previous task. In comparison of TCP and DCP based approaches to I_{full}, we observe that the former cases still yield comparable or better performance, however up to 50% pruning, again a more conservative result than that reported for the previous task.

4.4 Performance Comparison of Indexing Strategies: Best-in-Context Task

For the Best-in-Context task, we obtain the relevant documents exactly in the same way as in RiC. However, while ranking the documents, if the article node of the document is within these retrieved elements, we use its score as the document score. Otherwise, we use the score of the highest scoring retrieved element as the score of

Table 3. Effectiveness comparison of indexing strategies for BiC task

Indexing Strategy	Prune (%)	gP[0.05]	gP[0.10]	gP[0.25]	gP[0.50]	MAgP
I_{full}	0%	0.367	0.314	0.237	0.186	0.178
$I_{TCP, 0.3}$	30%	0.369	0.318	0.237	0.187	0.178
$I_{DCP, 0.3}$	30%	**0.388**	**0.332**	**0.246**	**0.187**	**0.184**
$I_{TCP, 0.5}$	50%	0.364	0.319	0.232	0.179	0.165
$I_{DCP, 0.5}$	50%	0.363	0.310	0.234	0.178	0.166
$I_{TCP, 0.7}$	70%	0.335	0.287	0.198	0.154	0.138
$I_{DCP, 0.7}$	70%	0.340	0.280	0.214	0.157	0.143
I_{direct}	65%	0.215	0.183	0.141	0.116	0.087
I_{direct}+$PROP_{SCORE}$	65%	0.127	0.132	0.120	0.100	0.086
I_{direct}+$PROP_{BM25}$	65%	0.156	0.151	0.136	0.115	0.074

this particular document. Then, we identify a best-entry-point (BEP) per document. In INEX 2008, a simple approach of setting the BEP as 1 is found to be very effective and ranked second among all participants [15]. Note that, this suggests starting to read each ranked document from the beginning. For our work, we also experimented with providing the offset of the highest scoring element per document as BEP [15], which yielded inferior results to the former approach. Thus, we only report the results where BEP is set to 1.

In Table 3, we compare indexing strategies in terms of the same evaluation metrics used in RiC task. As in RiC case, the performance obtained by using pruned index files with TCP and DCP is comparable to that of using the full element-index up to 50% pruning. At the 70% pruning level, both pruning approaches have losses in effectiveness with respect to I_{full}, but they are still considerably better than using I_{direct} with the (approximately) same index size. For instance, while MAgP for DCP is 0.143, the retrieval strategies using I_{direct} (with BM25), I_{direct}+$PROP_{SCORE}$ and I_{direct}+$PROP_{BM25}$ yield the MAgP figures of 0.087, 0.086 and 0.074, respectively. Again, basic retrieval using I_{direct} outperforms propagation based approaches, especially at the earlier ranks for generalized precision metric.

5 Conclusion

Previous experiences with XML collections suggest that element indexing is important for high performance in ad hoc retrieval tasks. In this study, we propose to use static index pruning techniques for reducing the size of a full element-index, which would otherwise be very large due to the nested structure of XML documents. We also compare the performance of term and document based pruning strategies to those approaches that use a direct element index that avoids indexing nested content more than once. Our experiments are conducted along the lines of previous INEX campaigns. The results reveal that pruned index files are comparable or even superior to the full element-index up to very high pruning levels for various ad hoc tasks (e.g., up to 70% pruning for Focused task and 50% pruning for RiC and BiC tasks) in terms of the retrieval effectiveness. Moreover, the performance of pruned index files is also better than that of the approaches using the direct index file at the same index size.

Future work directions involve extending our framework with some other static index pruning techniques and investigating the performance for the other query types (e.g., conjunctive and phrase queries, CAS queries, etc.).

Acknowledgments. This work is supported by The Scientific and Technological Research Council of Turkey (TÜBİTAK) by the grant number 108E008.

References

1. Altingovde, I.S., Ozcan, R., Ulusoy, Ö.: A practitioner's guide for static index pruning. In: Proc. of ECIR 2009, pp. 675–679 (2009)
2. Altingovde, I.S., Ozcan, R., Ulusoy, Ö.: Exploiting Query Views for Static Index Pruning in Web Search Engines. In: Proc. of CIKM 2009, pp. 1951–1954 (2009)
3. Blanco, R., Barreiro, A.: Boosting static pruning of inverted files. In: Proc. of SIGIR 2007, pp. 777–778 (2007)
4. Büttcher, S., Clarke, C.L.: A document-centric approach to static index pruning in text retrieval systems. In: Proc. of CIKM 2006, pp. 182–189 (2006)
5. Carmel, D., Cohen, D., Fagin, R., Farchi, E., Herscovici, M., Maarek, Y.S., Soffer, A.: Static index pruning for information retrieval systems. In: Proc of SIGIR 2001, pp. 43–50 (2001)
6. de Moura, E.S., Santos, C.F., Araujo, B.D., Silva, A.S., Calado, P., Nascimento, M.A.: Locality-Based pruning methods for web search. ACM TOIS 26(2), 1–28 (2008)
7. Denoyer, L., Gallinari, P.: The Wikipedia XML Corpus. SIGIR Forum 40(1), 64–69 (2006)
8. Garcia, S.: Search Engine Optimization Using Past Queries. Doctoral Thesis, RMIT (2007)
9. Geva, S.: GPX – Gardens Point XML Information Retrieval at INEX 2004. In: Fuhr, N., Lalmas, M., Malik, S., Szlávik, Z. (eds.) INEX 2004. LNCS, vol. 3493, pp. 211–223. Springer, Heidelberg (2005)
10. Geva, S.: GPX – Gardens Point XML IR at INEX 2005. In: Fuhr, N., Lalmas, M., Malik, S., Kazai, G. (eds.) INEX 2005. LNCS, vol. 3977, pp. 240–253. Springer, Heidelberg (2006)
11. Geva, S.: GPX – Gardens Point XML IR at INEX 2006. In: Proc. of INEX 2006 Workshop, pp. 137–150 (2006)
12. Guo, L., Shao, F., Botev, C., Shanmugasundaram, J.: XRANK: Ranked Keyword Search Over XML Documents. In: Proc. of the ACM SIGMOD 2003, pp. 16–27 (2003)
13. Initiative for the Evaluation of XML Retrieval (2009), `http://www.inex.otago.ac.nz/`
14. Itakura, K.Y., Clarke, C.L.A.: University of Waterloo at INEX 2008: Adhoc, Book, and Link-the-Wiki Tracks. In: Geva, S., Kamps, J., Trotman, A. (eds.) INEX 2008. LNCS, vol. 5631, pp. 132–139. Springer, Heidelberg (2009)
15. Kamps, J., Geva, S., Trotman, A., Woodley, A., Koolen, M.: Overview of the 2008 Ad Hoc Track. In: Geva, S., Kamps, J., Trotman, A. (eds.) INEX 2008. LNCS, vol. 5631, pp. 1–28. Springer, Heidelberg (2009)
16. Lalmas, M.: XML Retrieval. Morgan & Claypool, San Francisco (2009)
17. Ntoulas, A., Cho, J.: Pruning policies for two-tiered inverted index with correctness guarantee. In: Proc. of SIGIR 2007, pp. 191–198 (2007)

18. Sigurbjörnsson, B., Kamps, J.: The effect of structured queries and selective indexing on XML retrieval. In: Fuhr, N., Lalmas, M., Malik, S., Kazai, G. (eds.) INEX 2005. LNCS, vol. 3977, pp. 104–118. Springer, Heidelberg (2006)
19. Skobeltsyn, G., Junqueira, F., Plachouras, V., Baeza-Yates, R.: ResIn: a combination of results caching and index pruning for high-performance web search engines. In: Proc. of SIGIR 2008, pp. 131–138 (2008)
20. Su-Cheng, H., Chien-Sing, L.: Node Labeling Schemes in XML Query Optimization: A Survey and Trends. IETE Tech Rev 26, 88–100 (2009)
21. Zettair search engine (2009), `http://www.seg.rmit.edu.au/zettair/`
22. Zobel, J., Moffat, A.: Inverted files for text search engines. ACM Computing Surveys 38(2), 1–56 (2006)