Technical Report

No: BU-CE-1103

Combining Term and Document Popularities with Query Views for Static Index Pruning

Ismail Sengor ALTINGOVDE, Rifat OZCAN, and Özgür ULUSOY

February 2011

Bilkent University
Department of Computer Engineering
06800 Ankara, Turkey
http://www.cs.bilkent.edu.tr

Combining Term and Document Popularities with Query Views for Static Index Pruning

ISMAIL SENGOR ALTINGOVDE, RIFAT OZCAN, and
ÖZGÜR ULUSOY
Bilkent University

Static index pruning techniques permanently remove a presumably redundant part of an inverted file, to reduce the file size and query processing time. These techniques differ in deciding which parts of an index can be removed safely; i.e., without changing the top-ranked query results. As defined in the literature, the query view of a document is the set of query terms that access to this particular document, i.e., retrieves this document among its top results. In this study, we first propose using query views to improve the quality of the top results compared against the original results. We incorporate query views in a number of static pruning strategies, namely term-centric, document-centric, term popularity based and document access popularity based approaches, and show that the new strategies considerably outperform their counterparts especially for the higher levels of pruning and for both disjunctive and conjunctive query processing. Additionally, we combine the notions of term and document access popularity to form new pruning strategies, and further extend these strategies with the query views. The new strategies improve the result quality especially for the conjunctive query processing, which is the default and most common search mode of a search engine.

Categories and Subject Descriptors: Categories and Subject Descriptors: H.3.1 [**Information Storage and Retrieval**]: Content Analysis and Indexing *–indexing methods*; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval *–search process*

General Terms: Algorithms, Performance, Experimentation.

Additional Key Words and Phrases: Query view, static inverted index pruning.

1. INTRODUCTION

An inverted index is the state-of-the-art data structure for query processing in large scale information retrieval systems and Web search engines (SEs). In the last decades, several optimizations have been proposed to store and access inverted index files efficiently, while keeping the quality of the search relatively stable [Zobel and Moffat 2006]. One particular method is static index pruning, which aims to reduce the index file size and query execution time.

This research is supported by The Scientific and Technological Research Council of Turkey (TÜBİTAK) under the grant no 108E008.

Authors' addresses: I. S. Altingovde, R. Ozcan, and Ö. Ulusoy, Computer Engineering Department, Bilkent University, Ankara, 06800, Turkey; email: {ismaila, rozcan, oulusoy}@cs.bilkent.edu.tr.

The sole purpose of a static pruning strategy is staying loyal to the original ranking of the underlying search system for most queries, while reducing the index size, to the greatest extent possible. This is a non-trivial task, as it would be impossible to generate exactly the same results as produced by the *full* (unpruned) index for all possible queries. Therefore, pruning strategies attempt to provide quality guarantees for only top-ranked results, and try to keep in the pruned index those terms or documents that are the most important according to some measure, hoping that they would contribute to the future query outputs uttermost. The heuristics and measures used for deciding which items should be kept in the index and which of them should be pruned distinguish the static pruning strategies. Many proposals in the literature are solely based on the features of the collection and search system. For instance, in one of the pioneering works, Carmel et al. [2001] sort the postings in each term's list with respect to the search system's scoring function and remove those postings with the scores under a threshold. This is said to be a term-centric approach. In an alternative document-centric strategy, instead of considering posting lists, pruning is carried out for each document [Büttcher and Clarke 2006]. These two strategies, as well as some others reviewed in the next section essentially take into account the collection-wide features (such as term frequency) and search system features (such as scoring functions).

However, in the case of Web search, additional sources of information are also available that may enhance the pruning process and final result quality, which is the most crucial issue for search engines. In particular, query logs can serve as an invaluable source of information and provide further evidence for deciding which terms or documents should be kept in a pruned index to answer the future queries. A simple yet very effective index pruning strategy is based on the popularity of the terms in the queries, which can be determined from past query logs [Ntoulas and Cho 2007; Skobeltsyn et al. 2008; Baeza-Yates et al. 2007]. Clearly, such an approach depends on the frequency of the terms in the queries; however, it does not take into account the frequency of access to documents. In contrary, another approach proposed by Garcia [2007] exploits the notion of access popularity of documents, but neglects the term dimension. This latter pruning strategy is guided by the number of appearances of a document in the top results of the previous queries.

In the literature, for the purposes other than index pruning, query logs are also used to construct *query views*; i.e., a representation of a document in terms of the query terms [Castellanos 2003]. In the scope of our work, for a given document, all queries that rank this particular document among their top-ranked results constitute the query view of that document. For static pruning purposes, we exploit the query views in the following sense. We envision that, for a given document *d* and a term *t* in *d*, the appearance of *t* in *d*'s query view is the major evidence of its importance for *d*; i.e., it implies that *t* is a preferred way of accessing document *d* in the search system. Thus, any pruning strategy should avoid pruning the index entry <*d*> from the posting list of term *t* to the greatest extent possible.

In this study, our goal is to improve the quality[1] of the results obtained from a pruned index, which has vital importance for the SEs in a competitive market. Two key contributions of this work (besides many others as listed in the next section) are as follows. First, we propose to incorporate query views into the previous techniques that are using only collection and system features, like term- and document-centric strategies, as well as those that make use of query logs (i.e., to determine most popular terms and most frequently accessed documents, as discussed above). We show that, the pruning strategies coupled with the query views significantly improve the quality of the top-ranked results, especially at the higher levels of pruning.

Second, we combine the notions of term and document access popularity to form new pruning strategies, and further extend these strategies again with the query view idea. The new strategies improve the result quality especially for the conjunctive query processing, which is the default and most common search mode of a SE. For instance, such a combined strategy yields twice as good performance than solely using term popularity based pruning at a very high pruning level, namely, when 90% of the index is pruned.

1.1 Contributions

In addition to the key contributions mentioned above, our work provides a detailed comparison of several static index pruning approaches in the literature, proposes extensions to them and describes a realistic experimental framework. More concretely, the contributions and findings of this study are listed in detail as follows:

- *An exhaustive coverage of baseline static pruning approaches:* We fully explore the potential of previous pruning strategies, with special emphasis on the document access-based pruning. To this end, we provide an adaptive version of the term-centric pruning algorithm provided in [Garcia 2007]. We also introduce a new document-centric version of the access-based algorithm, and show that the latter outperforms its term-centric counterpart. Thus, with the addition of the term-centric

[1] In the scope of this work, by "result quality" we broadly mean the overlap between the results provided from the full (i.e., unpruned) index and from a pruned index.

[Carmel et al. 2001], document-centric [Büttcher and Clarke 2006] and term popularity based algorithms [Ntoulas and Cho 2007], we consider five baseline approaches in this study.

- *Query-view based static pruning approaches*: We couple the query view notion with all of these five pruning approaches. More specifically, the baseline algorithms are modified in such a way that the terms of a document that appear in the query view of this particular document are considered to be privileged and preserved in the index to the greatest possible extent during the pruning.
- *Evaluation of the pruning algorithms with and w/o query views:* We provide an effectiveness comparison of these baseline approaches (for their best performing setup in the literature) in a uniform framework for both disjunctive and conjunctive query processing; i.e., the most common query processing modes in SEs [de Moura et al. 2005]. To our knowledge, even this comparison alone adds value to the literature. Our experimental findings reveal that among the baseline strategies, the simple term popularity based method yields the best results for most of the cases; but document-centric version of the access-based algorithm can outperform it for conjunctive query processing at very high pruning levels. Then, of course, query view based approaches are compared with the baselines. We show that almost all strategies significantly benefit from the query views for the majority of the experiments.
- *Static pruning approaches that combine term popularity with document access popularity and query views:* As the term popularity and document access based approaches arise as the most competitive ones in our experiments, we propose new pruning algorithms that combine these two approaches. During pruning, the combined strategies first select the most popular terms to keep in the index, and then select their most popular documents, i.e., retrieved most frequently in top results of the queries. To our knowledge, there is no other strategy in the literature that combines these two dimensions, namely, term popularity and document access popularity. For the sake of completeness, we also combine term popularity strategy with term-centric and document-centric approaches, as sometimes used in the literature (e.g., [Ntoulas and Cho 2007; Skobeltsyn et al. 2008]). Furthermore, we incorporate query views into all of these strategies.

We also provide experiments to better understand the nature of term popularity and document access popularity. Our findings shed light on why combining both approaches can yield better results.

The rest of the paper is organized as follows. In the next section, we review the related work in the literature. In Section 3, we describe the baseline index pruning algorithms and in Section 4, we introduce the new pruning strategies that exploit the query views. Section 5 provides an experimental evaluation of all strategies in terms of top-ranked result quality. In Section 6, we first introduce new pruning strategies that combine the most important features as identified in our experiments; namely, term popularity, document access popularity and query views. Then, we provide an experimental comparison of the latter algorithms with several other combinations of pruning strategies. Finally, we summarize our main findings and point to future research directions in Section 7.

2. RELATED WORK

2.1 Static Inverted Index Pruning

In the last decade, a number of different approaches have been proposed for the static index pruning. In this study, as in [Büttcher and Clarke 2006], 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. Note that, this terminology is slightly different than that of [Carmel et al. 2001]. Additionally, we call a strategy *adaptive* if its pruning criteria (e.g., a threshold) dynamically changes for different terms or documents. In contrast, a *uniform* strategy applies pruning with a fixed threshold for all documents or terms.

In one of the earliest works in this field, Carmel et al. proposed term-centric approaches with uniform and adaptive versions [2001]. In this work, an idealized top-*k* pruning algorithm is introduced, which is guaranteed to generate the same answers (within an error of ε) as the original index for queries including less than $1/\varepsilon$ terms. It is observed that this idealized algorithm provides only negligible pruning effects, and thus it is relaxed by a score-shifting operation. After this modification, which also relaxes the theoretical guarantees, the adaptive version of 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. Roughly, adaptive top-*k* algorithm sorts the posting list of each term according to some scoring function (e.g., Smart's TF-IDF in [Carmel et al. 2001]) and removes those postings that have scores under a threshold determined for that particular term. In our study, this algorithm, which is referred to as *term-centric pruning (TCP)* strategy hereafter, is employed as a baseline pruning strategy and its further details are discussed in Section 3.1.

In [de Moura et al. 2005], the authors propose an index pruning approach that is tailored to support conjunctive and phrase queries, which requires a positional index. In this strategy, the term co-occurrence information is used to guide the pruning. In a nutshell, this strategy has three stages. First, the most significant sentences of the documents are determined. Next, these sentences are ranked and a fixed number of them are selected. Finally, the frequency and positional index files are constructed so that they only consider those terms and their positional information that appear in the selected sentences. In a follow-up work, a more sophisticated algorithm with the same goals is proposed [de Moura et al. 2008].

In [Blanco and Barreiro 2007b], another term-centric pruning strategy is suggested. In this work, the collection dependent stop-words are identified and totally removed from the index. To determine those terms to be pruned, several measures like inverse document frequency (*idf*), residual idf and term discriminative value are used. Their findings indicate that, although this approach can outperform the TCP strategy for some cases, the latter is better for short queries and obtaining high P@10 scores. This justifies our choice of TCP to be used in this work, as we essentially focus on improving the result quality for Web queries over a pruned index.

Another recently proposed term-centric pruning approach is based on the probability ranking principle [Blanco 2008]. Briefly, for each document in a term's posting list, this strategy computes a score that represents the significance of that term to the document, and prunes those that are below a global threshold. This approach is shown to be superior to TCP in terms of MAP results; however its performance for P@10 is less stable, but still comparable with TCP.

Finally, the access-based static pruning strategy discussed in [Garcia 2007] employs a query log and computes the number of appearances of each document in top-1000 results of the queries. These access-counts are then used to guide the pruning of posting lists for each term in the lexicon; i.e., in a term-centric fashion. This strategy is uniform, in the sense that for each term, a fixed number of postings that belong to the documents with highest access-count scores are stored in the pruned index, and the rest is pruned. In [Garcia 2007], the performance of this algorithm is shown to be somewhat discouraging, and as a remedy, the authors devise a mechanism to predict the query difficulty. Then, "simple" queries are processed by the pruned index whereas "difficult" ones are forwarded to the original index, which should also be stored. In this study, we provide an adaptive version of the term-centric approach outlined above. We also propose a document-centric version, which outperforms the former one. Further details of this approach are discussed in Section 3.2.

Note that, the access-based pruning approach is also adapted for dynamic pruning [Garcia et al. 2004; Garcia and Turpin 2006]. In that case, the query processing dynamically stops when a threshold is reached while processing a query term's posting list, which is sorted in access-count order. This approach is out of the scope of our work presented here and not elaborated further.

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 [Büttcher and Clarke 2006]. 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. However, the experimental setup in this latter work is significantly different than that of [Carmel et al. 2001]. That is, only the most frequent terms of the collection are pruned and the resulting (relatively small) index is kept in the memory, whereas the remaining unpruned body of index resides on the disk. During retrieval, if the query term is not found in the pruned index in memory, the unpruned index is consulted. In a more recent study [Altingovde et al. 2009a], 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 mostly outperforms DCP for various parameter selections. In this study, we also use the DCP strategy to prune the entire index, and employ it as one of the baseline strategies (see Section 3.1).

In most of the above works, it is either explicitly or implicitly assumed that the pruned index will replace the original one (e.g., at the back-end servers in a SE), and the pruning strategies are optimized for providing the most similar results to the original result. In this sense, these pruning approaches can be considered as *lossy*. In another line of research, it is proposed to use a pruned index while also keeping the full index at the back-end, so that the correctness of the queries can be always guaranteed. To this end, Ntoulas and Cho [2007] describe term and document pruning strategies with correctness guarantees. In Section 6, we further discuss the issues related to employing our pruning algorithms in such a lossless pruning architecture.

A similar approach is also taken in the ResIn framework [Skobeltsyn et al. 2008]. In ResIn, it is assumed that a pruned index is placed between the SE front-end and the broker, which is responsible for sending the queries to the back-end servers with the full index. In this case, the pruned index serves as a posting list cache, and the queries are passed to the broker and the back-end only when it is deduced that the query cannot be answered correctly. The originality of ResIn lies in its realistic architecture that also takes into account a dynamic result cache placed in front of the pruned index and the back-end. That is, all queries are filtered through the result cache, and only the misses are sent to the pruned index and/or back-end servers. Thus, the pruning algorithms employed in such an architecture should perform well essentially for the miss-queries. Their experiments show that term popularity based pruning serves well for the miss queries, whereas pruning lists (as in TCP) performs worse. A combination of both techniques is shown to provide a moderate increase in the hit rates, or equivalently, in the number of queries that can be answered correctly with the pruned index. In this work, we also consider a result cache and evaluate the pruning strategies using a test query stream that exhibits the same characteristics as the miss-queries of ResIn.

2.2 Query Views for Representing Documents

Query logs are exploited in several ways in the information retrieval literature. In the scope of this paper, we only focus on the related work for their usage as a representation model for documents. The concept of "query view" is first defined in [Castellanos 2003]. In this work, queries are used as features for modeling documents in a web site. Other works also use queries for document representation (called "query vector model") in the context of document selection algorithms for parallel information retrieval systems [Puppin et al. 2006; Puppin et al. 2010]. In this work, each query is associated with its top-*k* resulting documents and no click information is used. This is similar to our case, as we also restrict the notion of the query view only to the output of the underlying search engine and disregard the click through information. This choice makes sense for the purposes of pruning, as the aim of a static pruning algorithm is generating the same or most similar output with the underlying search system.

In a recent work [Poblete and Baeza-Yates 2008], the query log is mined to find "frequent query patterns" which form the "query-set model". Then each document is represented by the query-set model for clustering documents in a web site. This work suggests that query based representation dramatically improves the quality of the results. Another recent work [Antonellis et al. 2009] uses query terms as tags to label the documents that appear in the top-*k* results and are clicked by the users.

In our preliminary study [Altingovde et al. 2009b], we provided the first results for incorporating query views with four static index pruning algorithms (namely, TCP, DCP, aTCP and aDCP). Our current study significantly extends this prior work in many ways, such as the inclusion of popularity-based pruning (PP) algorithm, introduction of combined pruning algorithms, and an extensive experimental setup using various training and test query sets as well as an additional very large collection (i.e., ClueWeb09 Category B).

2.3 Other Mechanisms for Search Efficiency: Dynamic Pruning and Caching

While our focus in this study is on static index pruning, another complementary method for enhancing the search performance is dynamic pruning. These techniques do not remove any part of the index permanently; but aim to use only the most promising parts of posting lists during the query processing for increasing efficiency without deteriorating the retrieval effectiveness. For instance, *quit* and *continue* techniques as proposed by Moffat and Zobel [1996] enforce a limit on the number of accumulator entries that can be updated during query evaluation. This reduces the memory consumption for accumulators, which store partial similarities. Furthermore, these two strategies coupled with a skipping index are shown to improve Boolean and ranking-query efficiency. Similarly, a cluster-skipping index is proposed by Altingovde et al. [2008] to improve efficiency in a framework that includes document clusters. Persin et al. [1996] propose to use frequency-sorted indexes to avoid reading entire posting lists from the disk and then processing them. More recently, Anh et al. [2001] introduce the impact-sorted indexes and dynamic pruning techniques that operate on top of this data structure. In this approach, posting lists are reorganized so that blocks of higher impact documents are stored first. The integration of proximity weighting models into dynamic pruning techniques is discussed in [Tonellotto et al. 2010].

An orthogonal mechanism to pruning is caching, which can totally eliminate the cost of query processing (i.e., by result caching), or significantly reduce it (i.e., by list caching). The former case is important, because a result cache can significantly alter the properties of a query stream that is directed to a pruned index, as discussed above. The latter case is also important, as the pruned index can indeed serve as a list cache [Skobeltsyn et al. 2008]. In the literature, the term based pruning mechanism of [Ntoulas and Cho 2007] is also used for filling a list cache in [Baeza-Yates et al. 2007].

3. STATIC PRUNING APPROACHES

We start with describing how exactly TCP and DCP algorithms are implemented in our framework. Next, we describe access-based TCP, as a slightly modified version of Garcia’s uniform pruning algorithm [2007]. Then we introduce a document-centric version of the latter strategy. As a final baseline strategy, we discuss term popularity based pruning.

3.1 Static Pruning Strategies Exploiting Collection and Search System Features

Term-Centric Pruning (TCP) strategy. As it is mentioned in the previous section, TCP, the adaptive version of the top-k algorithm proposed in [Carmel et al. 2001], is reported to be very successful in static pruning especially for disjunctive processing of the queries. 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. As in [Blanco and Barreiro 2007b], we disregard any theoretical guarantees and determine ε values according to the desired pruning level.

Following the recommendations in a recent study [Blanco and Barreiro 2007a], we employ BM25 as the scoring function for TCP and entirely discard the terms with document frequency $f_t > N/2$ (where N is the total number of documents) as their BM25 score turns out to be negative. Figure 1 shows the TCP strategy as adapted in our setup.

```
Algorithm Term-Centric Pruning (TCP)
Input: I, k, ε, N
for each term t in I
     fetch the postings list I_t from I
     if |I_t| > N / 2
          remove I_t entirely from I
     if |I_t| > k
        for each posting entry <d>,
            compute Score(t, d) with BM25
        let z_t be the k^th highest score among the scores
        τ_t ← z_t * ε
        for each posting entry <d>
            if Score(t, d) ≤ τ_t
                  remove entry <d> from I_t
```

Fig. 1. Pseudocode for term-centric pruning (TCP).

Document-Centric Pruning (DCP) strategy. In this study, we apply the DCP strategy for the entire index, which is slightly different than pruning only the most frequent terms as originally proposed by [Büttcher and Clarke 2006]. Additionally, instead of scoring each term of a document with KLD, we prefer to use BM25, to be compatible with TCP. In a recent work, BM25 is reported to perform better than KLD for DCP [Altingovde et al. 2009a]. Finally, in [Büttcher and Clarke 2006] it is again shown that the uniform strategy; i.e., pruning a fixed number of terms from each document, is inferior to the adaptive strategy, where a fraction (λ) of the total number of unique terms in a document is pruned. Figure 2 shows the algorithm for the DCP strategy.

Algorithm *Document-Centric Pruning (DCP)*

Input: D, λ
1: **for** each document $d \in D$
2: sort $t \in d$ in descending order w.r.t. Score(d, t)
3: remove the last $|d|*\lambda$ terms from d

Fig. 2. Pseudocode for document-centric pruning (DCP).

3.2 Static Pruning Strategies Exploiting Previous Query Logs

Access-based Term-Centric Pruning (aTCP) strategy. In the literature, the strategy of Garcia [2007] is one of the earliest works that use the search engine query logs to guide the static index pruning process. However, this work does not use the actual content of the queries, but just makes use of the access count of a document; i.e., the number of times a document appears in top-*k* results of queries, where *k* is set to 1000. The proposed static pruning algorithm applies the, so-called, MAXPOST heuristic, which simply keeps a fixed number of postings with the highest access counts in each term's posting list.

The result of the MAXPOST approach is not very encouraging. Despite considerable gains (up to 75%) in the query processing time, the reduction in accuracy is significant; i.e., up to 22% drop in MAP is observed when only 35% of the index is pruned (see p.114, Figure 5.2 in [Garcia 2007]). We attribute this result to the uniform pruning heuristic, which is shown to be a relatively unsuccessful approach for other strategies (e.g., TCP and DCP) as discussed above.

For this study, we decide to implement an adaptive version of the MAXPOST approach. Since it iterates over each term and removes some postings, we classify this approach as term-centric, and call the adaptive version *access-based TCP* (aTCP). In this case, instead of keeping a fixed number of postings in each list, we keep a fraction (μ) of the number of postings in each list. Figure 3 shows aTCP strategy.

Algorithm *Access-based Term-Centric Pruning (aTCP)*

Input: I, μ, AccessScore[]
1: **for** each term t in I
2: fetch the postings list I_t from I
3: sort $d \in I_t$ in descending order w.r.t. AccessScore[d]
4: remove the last $|I_t|*\mu$ postings from I_t

Fig. 3. Pseudocode for access-based term-centric pruning (aTCP).

Access-based Document-Centric Pruning (aDCP) strategy. In this study, we propose a new access-based strategy. Instead of pruning the postings from each list, we propose to prune documents entirely from the collection, starting from the documents with the smallest access counts. The algorithm is adaptive in that, for an input pruning fraction (μ), the pruning iterates while the total length of pruned documents is less than $|D|*\mu$, where $|D|$ is the collection length; i.e., sum of the number of unique terms in each document. Figure 4 presents this strategy, which we call *access-based DCP* (aDCP).

Note that, for both of the access-based approaches (aTCP and aDCP) many documents may have the same access count. To break the ties, we need a secondary key to sort these documents. In this study, we simply use the URL of the Web pages and sort those documents with the same access count in lexicographical order. It is also possible to consider the length of a document or document URL, which are left as a future work.

Algorithm *Access-based Document-Centric Pruning (aDCP)*

```
Input: D, μ, AccessScore[]
sort d ∈ D in descending order w.r.t. AccessScore[d]
NumPrunedPostings ← 0
while NumPrunedPostings < |D|*μ
    remove the document d with the smallest access score
    NumPrunedPostings ← NumPrunedPostings + |d|
```

Fig. 4. Pseudocode for access-based document-centric pruning (TCP).

Popularity-based Pruning (PP) strategy. This is a simple yet very effective pruning strategy employed in the previous studies[2] for the purposes of index pruning and caching [Ntoulas and Cho 2007; Skobeltsyn et al. 2008; Baeza-Yates et al. 2007]. In this method, the terms that appear in the query log are sorted in descending order of the *term gain* score. Term gain score is simply computed as the ratio of the total number of queries that include a term t (popularity(t)) to the length of this term's posting list ($|I_t|$). Then, we keep the posting lists of the terms with the highest gain scores so that the total size of these lists does not exceed the required size of the pruned index. In Figure 5, this greedy strategy is shown. For the purposes of presentation, we assume that each term t in the index is associated with a value P_t, which is set to 1 if t would be pruned, and 0 otherwise. For an input pruning fraction (μ), the algorithm iterates over term gain score sorted list L of terms and sets P_t to 1, as long as the total length of these lists is less than $|I|*(1-\mu)$, where $|I|$ is the index size, i.e., sum of the lengths of all posting lists.

[2] Note that, this strategy is also called as keyword or term pruning in some of the previous works. Here, to prevent confusion with TCP, we call it popularity-based pruning.

Algorithm *Popularity-based Pruning (PP)*

```
Input: I, μ, popularity[]
L ← sort terms in I the descending order of TermGain(t) = popularity[t]/|I_t|
NumRemainingPostings ← 0
∀t P_t ← 0
while NumRemainingPostings < |I| * (1-μ)
    extract term t with the highest gain from L
    P_t ← 1
    NumRemainingPostings ← NumRemainingPostings + |I_t|
for each term t in I
    if P_t = 0
        remove I_t from I
```

Fig. 5. Pseudocode for popularity-based pruning (PP).

4. STATIC INDEX PRUNING USING QUERY VIEWS

In this section, we first define the notion of query view (*QV*) for a document, and then introduce the pruning strategies that incorporate the query view heuristic. Let's assume a document collection $D = \{d_1,\ldots,d_N\}$ and a query log $Q = \{Q_1, \ldots, Q_M\}$, where $Q_i = \{t_1,\ldots,t_q\}$. After this query log Q is executed over D, the top-k documents (at most) are retrieved for each query Q_i, which is denoted as $R_{Qi,k}$. Now, we define the query view of a document d as follows:

$$QV_d = \cup\, Q_i, \text{ where } d \in R_{Qi,k}$$

That is, each document is associated with a set of terms that appear in the queries which have retrieved this document within the top-k results. Without loss of generality, we assume that during the construction of the query views, queries in the log are executed in the conjunctive mode; i.e., all terms that appear in the query view of a document also appear in the document.

The set of query views for all documents, QV_D, can be computed efficiently either offline or online. In an offline computation mode, the search engine can execute a relatively small number of queries on the collection and retrieve, say, top-1000 results per query. Note that, as discussed in [Garcia and Turpin 2006], it may not be necessary to use all of the previous log files; the most recent log and/or sampling from the earlier logs can be sufficiently representative. In Section 5, we show that even small query logs (e.g., of 10K queries with top-1000 results) provide gains in terms of effectiveness. On the other hand, in the online mode, each time a query response is computed, say, top-10 results (i.e., only document ids) for this query can also be stored in the broker (or, sent to a dedicated query view server). Note that, such a query view server can store results for millions of queries in its secondary storage to be used during the index pruning, which is actually an offline process. In the experiments, we also provide the effectiveness figures obtained for the query views that are created by using only top-10 results.

We exploit the notion of query views for static index pruning, as follows. We envision that for a given document, the terms that appear as query terms to rank this document within top results of these queries should be privileged, and should not be pruned to the greatest extent possible. That is, as long as the target pruned index size is larger than the total query view size, all query view entries are kept in the index. In what follows, we introduce five pruning strategies that exploit the query views, based on the TCP, DCP, aTCP, aDCP and PP strategies, respectively.

Term-Centric Pruning with Query Views (TCP-QV). This strategy is based on TCP, but employs query views during pruning. In particular, once the pruning threshold (τ_t) is determined for a term *t*'s posting list, the postings that have scores below the threshold are not directly pruned. That is, given a posting *d* in the list of term *t*, if $t \in QV_d$, this posting is preserved in the index, regardless of its score. This modification is presented in Figure 6. Note that, by only modifying line 11, the query view heuristic is taken into account to guide the pruning.

Algorithm *Term-Centric Pruning with Query Views (TCP-QV)*

Input: $I, k, \varepsilon, N, QV_D$
1-9: *// The same as Figure 1 and not repeated here to save space*
10: **for** each posting entry <*d*>,
11: **if** Score(t, d) $\leq \tau_t$ **and** $t \notin QV_d$
12: remove entry <*d*> from I_t

Fig. 6. Pseudocode for term-centric pruning with query views (TCP-QV).

Document-Centric Pruning with Query Views (DCP-QV). In this case, for the purpose of discussion, let's assume that each term *t* in a document *d* is associated with a priority score Pr_t, which is set to 1 if $t \in QV_d$ and 0 otherwise. The terms of a document *d* are now sorted (in descending order) according to these two keys, first the priority score and then score function output. During the pruning, last $|d|*\lambda$ terms are removed, as before. This strategy is demonstrated in Figure 7.

***Algorithm** Document-Centric Pruning with Query Views (DCP-QV)*

Input: D, λ, QV_D

for each document d in D
    for each term $t \in d$
        if $t \in QV_d$ **then** $Pr_t \leftarrow 1$ **else** $Pr_t \leftarrow 0$
    sort $t \in d$ in descending order wrt. first Pr_t, then Score(d, t)
    remove the last $|d|*\lambda$ terms from d

Fig. 7. Pseudocode for document-centric pruning with query views (DCP-QV).

Access-based Term-Centric Pruning (aTCP) with Query Views (aTCP-QV). In aTCP strategy, again for the purposes of discussion, we assume that each posting d in the list of a term t is associated with a priority score Pr_d, which is set to 1 if $t \in QV_d$ and 0 otherwise. Then, the postings in the list are sorted in the descending order of the two keys, first the priority score and then the access count. During the pruning, last $|I_t|*\mu$ postings are removed (Figure 8).

***Algorithm** Access-based Term-Centric Pruning with QV (aTCP-QV)*

Input: I, μ, AccessScore[], QV_D

for each term t in I
    fetch the postings list I_t from I
    for each posting entry <d>,
        if $t \in QV_d$ **then** $Pr_d \leftarrow 1$ **else** $Pr_d \leftarrow 0$
    sort $d \in I_t$ in desc. order w.r.t. first Pr_d then AccessScore[d]
    remove the last $|I_t|*\mu$ postings from I_t

Fig. 8. Pseudocode for access-based term-centric pruning with query views (aTCP-QV).

Access-based Document-Centric Pruning (aDCP) with Query Views (aDCP-QV). In this case, we again prune the documents starting from those with the smallest access counts until the pruning threshold μ is reached. But, while pruning documents, those terms that appear in the query view of these documents are kept in the index. This is shown in Figure 9.

Note that, for all algorithms described in this section, if the desired size of the pruned index is less than the total size of the query views ($|QV_D|$) it is obligatory to prune some of the postings that appear in the query views, as well. In this case, we first remove all posting that are not in the query views, and then apply the original algorithm (i.e., either one of TCP, DCP, aTCP or aDCP) to the remaining postings. In effect, we apply the original algorithm over the index that only includes postings in QV_D. This stage is not shown in the algorithms for the sake of simplicity.

```
Algorithm Access-based Document-Centric Pruning with QV (aDCP-QV)
Input: D, μ, AccessScore[], QV_D
sort d ∈ D in descending order w.r.t. AccessScore[d]
NumPrunedPostings ← 0
while NumPrunedPostings < |D|*μ
    fetch d with the smallest score
    for each term t ∈ d
        if t ∉ QV_d
            remove t from d
            NumPrunedPostings ← NumPrunedPostings + 1
```

Fig. 9. Pseudocode for access-based document-centric pruning with query-views.

Popularity-based Pruning (PP) strategy with Query Views (PP-QV). In this strategy, again starting from the terms with the highest gain scores, we first attempt to keep the query view of each term in the pruned index. If all postings in QV_D are stored in the pruned index and there is still some space available (i.e., $|QV_D| < |I|*\mu$), then we make another pass on terms again in descending order of gain scores (lines 8-11 in Figure 10). This second pass aims to keep the full lists of the terms with highest gain scores, instead

```
Algorithm Popularity-based Pruning with QV (PP-QV)
Input: I, μ, popularity[]
1:  L ← sort terms in I the descending order of TermGain(t) = popularity[t]/|I_t|
2:  ∀t P_t ← 0, Q_t ← 0
3:  NumRemainingPostings ← 0
4:  while NumRemainingPostings < |I| * (1-μ) and L is not empty
5:      extract term t with the highest gain from L
6:      NumRemainingPostings ← NumRemainingPostings + |QV_t|
7:      Q_t ← 1
8:  reset L as in line (1)
9:  while NumRemainingPostings < |I| * (1-μ)
10:     extract term t with the highest gain from L
11:     NumRemainingPostings ← NumRemainingPostings + (|I_t|- |QV_t|)
12:     P_t ← 1
13: for each term t in I
14:      if P_t = 0 and Q_t = 0
15:          remove I_t from I
16:      else if P_t = 0 and Q_t = 1
17:          remove I_t-QV_t from I
```

Fig. 10. Pseudocode for popularity-based pruning with query views (PP-QV).

of solely keeping the query views, till the desired size of the pruned index is reached. This approach is presented in Figure 10. As in PP (Figure 5), P_t indicates whether the full list of term t would be pruned, or not. Q_t indicates whether the query view of the term t would be pruned or not. QV_t denotes all postings <d> in I_t such that $t \in QV_d$.

As a final remark, in Figures 6 to 10, we show the use of query views in a simplistic manner for the purposes of discussion, without considering the actual implementation. For instance, for TCP-QV case, it would be more efficient to first create an inverted index of the QV_D and then process the original index and query view index together; i.e., in a merge-join fashion, for each term in the vocabulary. We presume that for all five approaches employing query views, the additional cost of accessing an auxiliary data structure for QV_D (either the actual or inverted data) would be reasonable, given that the query terms highly overlap and only a fraction of documents in the collection have high access frequency [Garcia et al. 2004]. Furthermore, it is not necessary to use all previous query logs, as discussed above [Garcia and Turpin 2006]. That is, the size of these data structures would be much smaller when compared to the actual collection; i.e., Web.

5. EXPERIMENTAL EVALUATION

5.1 Experimental Setup

Document collection and indexing. For this study, we obtained the list of URLs that are categorized at the Open Directory Project (ODP) Web directory (www.dmoz.org). Among these links, we successfully downloaded around 2.2 million pages, which take 37 GBs of disk space in uncompressed HTML format. This ODP dataset constitutes our primary document collection for this study. Additionally, we use a second and larger collection, namely ClueWeb09-B [Clarke et al. 2010], for a subset of experiments involving the best performing pruning strategies.

We indexed both datasets using the publicly available Zettair IR system (www.seg.rmit.edu.au/zettair/). During the indexing, Zettair is executed with the "no stemming" option. All stop-words and numbers are included in the index, yielding vocabularies of around 20 and 160 million terms for ODP and ClueWeb09-B collections, respectively. Once the initial indexes are generated, we used our homemade IR system to create the pruned index files and execute the training and test queries over them. The resulting index files are document-level, i.e., each posting involves document identifier and term frequency fields (adding up to 8 bytes per posting).

Query log normalization. We use a subset of the AOL Query Log (http://imdc.datcat.org/collection/1-003M-5) that contains 20 million queries of about 650K people for a period of 12 weeks. The query terms are normalized by case-folding, sorting in the alphabetical order and removing the punctuation and stop-words. We consider only those queries, of which all terms appear in the collection vocabularies. This restriction is forced to guarantee that the selected queries are sensible for the datasets.

Training and test query sets. From the normalized query log subset, we construct training and test sets. Training query sets are used to compute the term popularities as well as the access counts and query views for the documents, and they are created from the first half (i.e., 6 weeks) of the log. The test sets that are used to evaluate the performance for different pruning strategies are constructed from the second half (last 6 weeks) of the log. During the query processing with both training and test sets, a version of BM25 scoring function as described in [Büttcher and Clarke 2006], is used.

In the training stage, queries are executed in the conjunctive mode and top-*k* results per query are retrieved. To observe the impact of the training set size, we created training sets of 10K, 50K, 518K and 1.8M distinct queries that are selected randomly from the first half of the log, and obtained top-1000 results per query. To further investigate the impact of the result set size, namely, *k*, we obtained only top-10 results for the latter two training sets (i.e., including 518K and 1.8M queries). Thus, we have six different training query logs with varying number of queries and results per query. These training sets are executed on the ODP dataset. Characteristics of the training sets are provided in Table I.

Table I. Characteristics of the training query sets (wrt. the ODP collection)

	10K-top1000	**50K-top1000**	**518K-top1000**	**1.8M-top1000**	**518K-top10**	**1.8M-top10**
Access %	30%	54%	79%	85%	33%	50%
QV Size (%)	35MB (1%)	143MB (4%)	647MB (20%)	1,093MB (34%)	53MB (2%)	148MB (5%)

In the first row of Table I, we provide the access percentage achieved by each training set; i.e., the percentage of documents that appear at least once in a query result. In the second row of the table, we report the total size of the query views ($|QV_D|$), which is the sum of the number of unique query terms that access to each document. We also provide the ratio of $|QV_D|$ to the ODP collection size (|D|). Both values increase as the number of queries increase, however the increments follow a sub-linear trend. This is due to the heavy-tailed distribution of accesses to documents as shown before [Garcia 2007].

Remarkably, access percentages for 10K-top1000 and 518K-top10 training sets are very close, which imply that access counts and query views with similar characteristics can be either obtained by using a relatively small query log and larger number of results, or using a larger query log but retrieving smaller number of, say only top-10, results. The former option can be preferred during an offline computation, whereas the latter can be achieved for an online computation. For instance, a search engine can store the top-10 document identifiers per query (maybe at a dedicated server) on the fly to easily compute the query views when required. Note that, these observations are also valid for the 50K-top1000 vs. 1.8M-top10 sets. In the experiments, we show that these sets also yield relatively similar effectiveness figures.

For the majority of the experiments reported in the next section, we use a test set of 1000 randomly selected queries from the second half of the AOL log. These queries are normalized as discussed above. We keep only those queries that can retrieve at least one document from our collections when processed in the conjunctive mode. By definition, the test set is temporally disjoint from the training sets. Furthermore, we guarantee that train and test sets are query-wise disjoint by removing all queries from the test set that also appear in the training sets (after the normalization stage). But, some of the terms in the queries in both sets, of course, may overlap. This set is referred to as test-1000 in the following sections.

Furthermore, our test-1000 set includes only singleton queries that appear only once in the query log. In this sense, our test set is similar to the miss-queries as described by the ResIn architecture [Skobeltsyn et al. 2008]; i.e., those queries that cannot be found in the result-cache and should be forwarded to the pruned index. In fact, we observed that our test set exhibits the same characteristics of the miss-queries as reported in ResIn (see Figure 3 in [Skobeltsyn et al. 2008]). This means that, the algorithms discussed in this study are evaluated using a test set of queries that realistically represent the query stream sent to a pruned index in a typical SE setup.

Compatibility of the dataset and query sets. As discussed in [Webber and Moffat 2005], the compatibility of the query log and underlying document collection is a crucial issue for the reliability of an experimental framework. Intuitively, we consider that our datasets and query log are compatible, since both ODP and ClueWeb datasets include general Web pages and AOL log is a general search engine log. We further investigated and demonstrated the compatibility of ODP dataset and AOL queries in another study [Ozcan et al. 2011].

Evaluation measure. In this work, we compare the top-k results obtained from the original index against the pruned index, where k is 10 (the results for k =2, 100 and 1000 reveal similar trends and are not reported here to save space). To this end, we employ the symmetric difference measure as discussed in [Carmel et al. 2001]. That is, for two top-k

lists, if the size of their union is y and the size of their symmetric difference is x, symmetric difference score $s = 1- x/y$. The score of 1 means exact overlap, whereas the score of 0 implies that two lists are disjoint. The average symmetric difference score is computed over the individual scores of 1000 test queries and reported in the following experiments.

Note that, it is also possible to use standard IR metrics (such as P@10 or MAP) for evaluating pruned results considering the results obtained from the full index as the ground-truth (as in [Garcia 2007]). We observed that both metrics (as computed over top-10 results) yield exactly the same trends with symmetric difference score measure for comparing the pruning algorithms, but absolute scores for traditional metrics are slightly higher. In this work, we only report symmetric difference scores, whereas MAP and P@10 results are discussed in Appendix 3.

Parameters for the pruning strategies. The pruned index files are obtained at the pruning levels ranging from 10% to 90% (with a step value of 10%) by tuning the ε, λ and μ parameters in the corresponding algorithms. All index sizes are considered in terms of their raw (uncompressed) sizes. For TCP, top-*k* parameter is set to 10 during pruning.

5.2 Results

Statistical significance of the results. All results obtained over the ODP collection, unless stated otherwise, are tested for statistical significance at 0.05 level. In particular, for the results in Tables II-V and Figures 11-12; at each pruning level, the output of 1000 test queries for a baseline algorithm and its query view based counterpart are compared using the paired t-test and Wilcoxon signed rank test. For Tables II and III, all improvements greater than 1% are statistically significant. For Tables IV and V, almost all improvements (except two cases in Table V) are significant. For the plots in Figures 11 and 12, there are only a few cases where a query view based strategy yields no significant improvements (especially for smaller training sets) and these cases are discussed in text as the space permits. Additionally, for the results shown in Tables II-V, we made a one-way ANOVA analysis (followed by Tukey's post hoc test) among the i) all baseline strategies, and ii) all query view based strategies, separately. This is because we compare the performance of the baseline (or, query view based) algorithms among each other to see which one is the most appropriate for certain cases. In the textual discussions, we mostly refer to the findings for which differences are found to be statistically significant.

In what follows, we analyze how the query views improve the performance of the baseline strategies for disjunctive and conjunctive query processing modes. Besides, we also investigate the effects of different training query set sizes on the algorithms that make use of a query log. All of the experiments reported in the rest of this section are conducted on the ODP collection.

Performance of the query views: disjunctive mode. In Figure 11, we provide the average symmetric difference scores for retrieving top-10 results in disjunctive query processing mode (please see Appendix 1 for the corresponding results in tabular format). In all plots, it is clear that query-view based strategies considerably improve performance of the corresponding baseline algorithms. In Figures 11(a) and 11(b), we see that query-views almost double the effectiveness of the respective baseline algorithms TCP and DCP, especially for the higher levels of pruning. It is also seen that the effectiveness of TCP-QV and DCP-QV improves proportionally to the training set size; i.e., higher performance is obtained for larger training sets. Still, even a training set of 10K queries improves performance in a statistically significant manner.

For the access and term popularity based strategies, to simplify the plots, we only provide the performance for training sets of 1.8M queries with top-10 and 1000 results (Figures 11(c) to 11(e)). For all three algorithms –namely aTCP, aDCP and PP, the query view based strategies outperform their counterparts for these training sets. Interestingly, the effectiveness figures of aTCP and aDCP are higher for the training sets using top-10 results. This means that for those strategies that actually make use of evidences from a query log, using a large result set (e.g., 1000) is not beneficial and indeed, may mislead the decision mechanism of the algorithms.

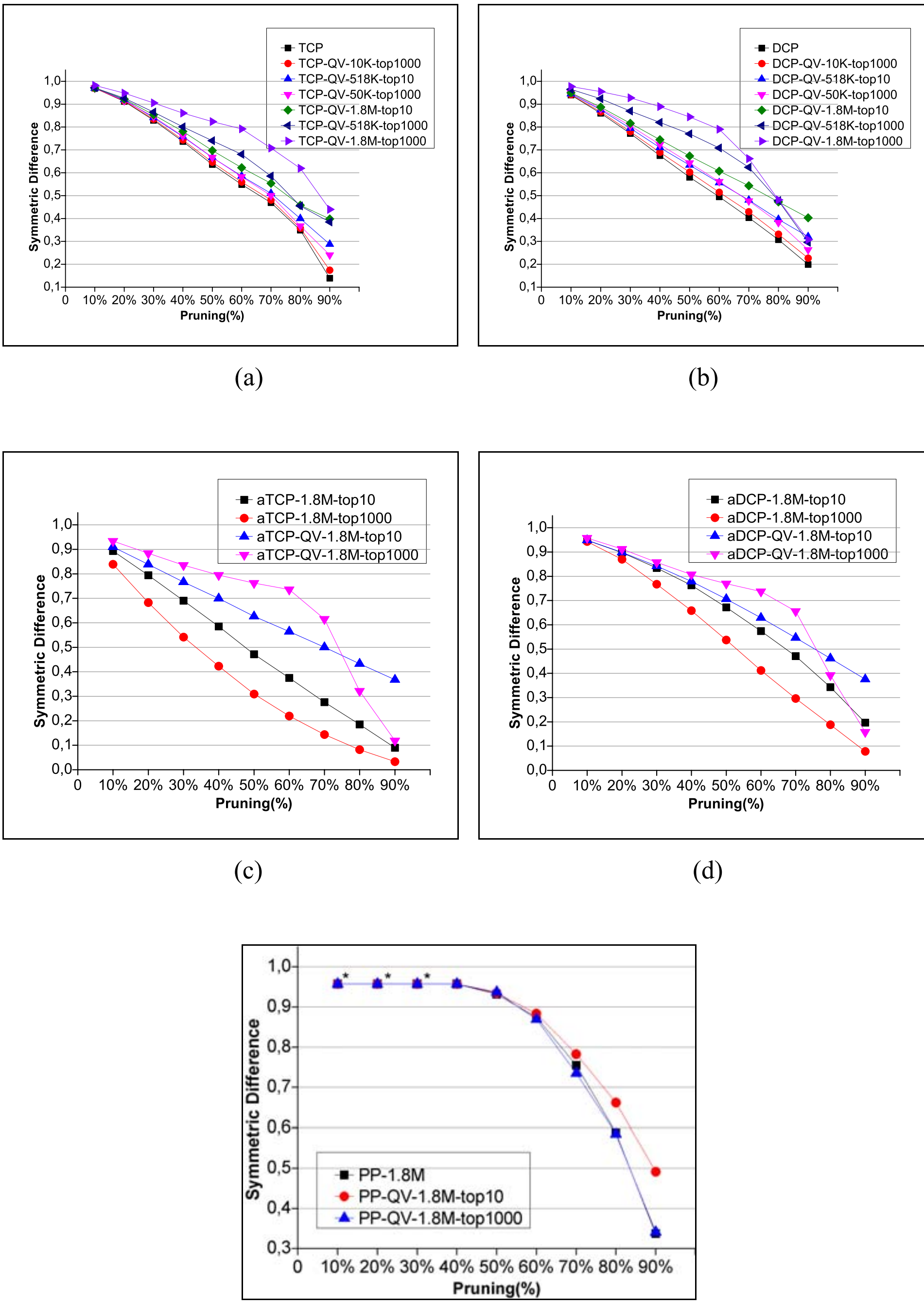

Fig. 11. Retrieval performance of index pruning strategies on the ODP collection using different training sets in disjunctive querying mode: (a) TCP vs. TCP-QV, (b) DCP vs. DCP-QV, (c) aTCP vs. aTCP-QV, (d) aDCP vs. aDCP-QV and (e) PP vs. PP-QV.

Note that, in Figures 11(a) to 11(d), we also observe that the performance of an algorithm trained with 1.8M-top1000 set gets worse than an algorithm trained with the same set but top-10 results, especially after 70% pruning level. We explain this phenomenon as follows. For the 1.8M-top1000 training set, the size of the query view is as large as 34% of the index (see Table I); so up to 70% pruning, we don't prune any postings that are included in the query views. After this point, we start pruning the query views using the original logic of the underlying algorithm (as discussed in Section 4), and observe a sharp decrease in the effectiveness figures. On the other hand, for top-10 case, the query view size is much smaller (i.e., around 5% of the index) and thus we never prune the query views. This implies that, a large query view provides benefits when there is enough space to fit it in (as in the case of 50% pruning level). Otherwise, it is better to obtain a more compact query view initially, say, using top-10 results only; rather than constructing a larger query view and pruning it afterwards. Furthermore, while pruning the query view, it may be more useful to devise a specific algorithm for this purpose, instead of using the original pruning algorithm. For instance, in addition to using the appearance of a term in the query view of a document, it is possible to exploit its access frequency; i.e., number of times a document is accessed by a query including that particular term. Note that, this is different than the access-based pruning discussed before. We leave exploring this direction as a future work.

Finally, in Figure 11(e), we discuss our findings for the PP and PP-QV algorithms. For these algorithms, posting lists of all terms that appear in the training queries correspond to 60% of the original index. When only these terms are kept in the index, i.e., at 40% pruning, the algorithms yield a very high effectiveness score (96%). Thus, the actual pruning is effective after this level, and for smaller pruning levels (between 10%-30%), we simply repeat the value observed at 40%. The repeated values for these pruning levels are shown with asterisks in Figures 11(e) and 12(e); and italicized in Tables II-III. Another approach could be filling the remaining available space with the lists of the randomly selected terms that don't appear in the training queries. We expect that this would only slightly improve the performance, which is already very high at 40% as discussed above, and do not take this path in this paper.

Our experiments reveal that query view also has the potential to improve the PP algorithm, which is the most practical approach that can be used for pruning and caching at search engines [Skobeltsyn et al. 2008; Baeza-Yates et al. 2007]. For this case, training with 1.8M-top1000 set does not yield any significant changes in the effectiveness. The gains are more emphasized especially at higher pruning levels (i.e., when more than 70% of the index is pruned) and using 1.8M-top10 training set.

Performance of the query views: conjunctive mode. In Figure 12, we demonstrate the behavior of the algorithms for the conjunctive processing mode (please see Appendix 1 for the corresponding results in tabular format). Again, TCP-QV and DCP-QV achieve higher scores when larger number of queries and top-1000 results are used during the training (Figures 12(a) and 12(b)). As before, for the highest pruning levels (i.e., more than 70%), using top-10 results is better than using top-1000 for the same query set. Nevertheless, query views improve the performance in all cases.

For aDCP and aTCP, trends are also similar to disjunctive case; but for their query view based counterparts, now using 1.8M-top1000 queries is better than the 1.8M-top10 set (see Figures 12(c) and 12(d)). For instance, for aDCP-QV algorithm, the training set 1.8M-top10 does not yield significantly different results (as also seen from the overlapping lines in Figure 12(d)). In this case, the improvements with query views are obtained when top-1000 results are used during the training.

Finally, in Figure 12(e), we demonstrate the performance of PP in comparison to PP-QV. Again, PP-QV yields significant improvements at very high pruning percentages and when 1.8M-top10 training set is employed.

An overall comparison of the pruning algorithms: disjunctive mode. In this part, we discuss and compare the performance of different pruning strategies in more detail. In Table II, we provide average symmetric difference results of all of the pruning strategies for the top-10 results and disjunctive query processing mode (on the ODP collection). For those strategies that make use of a query log –namely, aDCP, aTCP, PP and all query view based strategies; we employed our largest training set with top-10 results, i.e., 1.8M-top10 set. This is a realistic choice, because for a real search engine it would be practical to store top-10 document identifiers for a large number of queries. Furthermore, in above plots (Figure 11 and 12), this training set yields improvements for most of the cases.

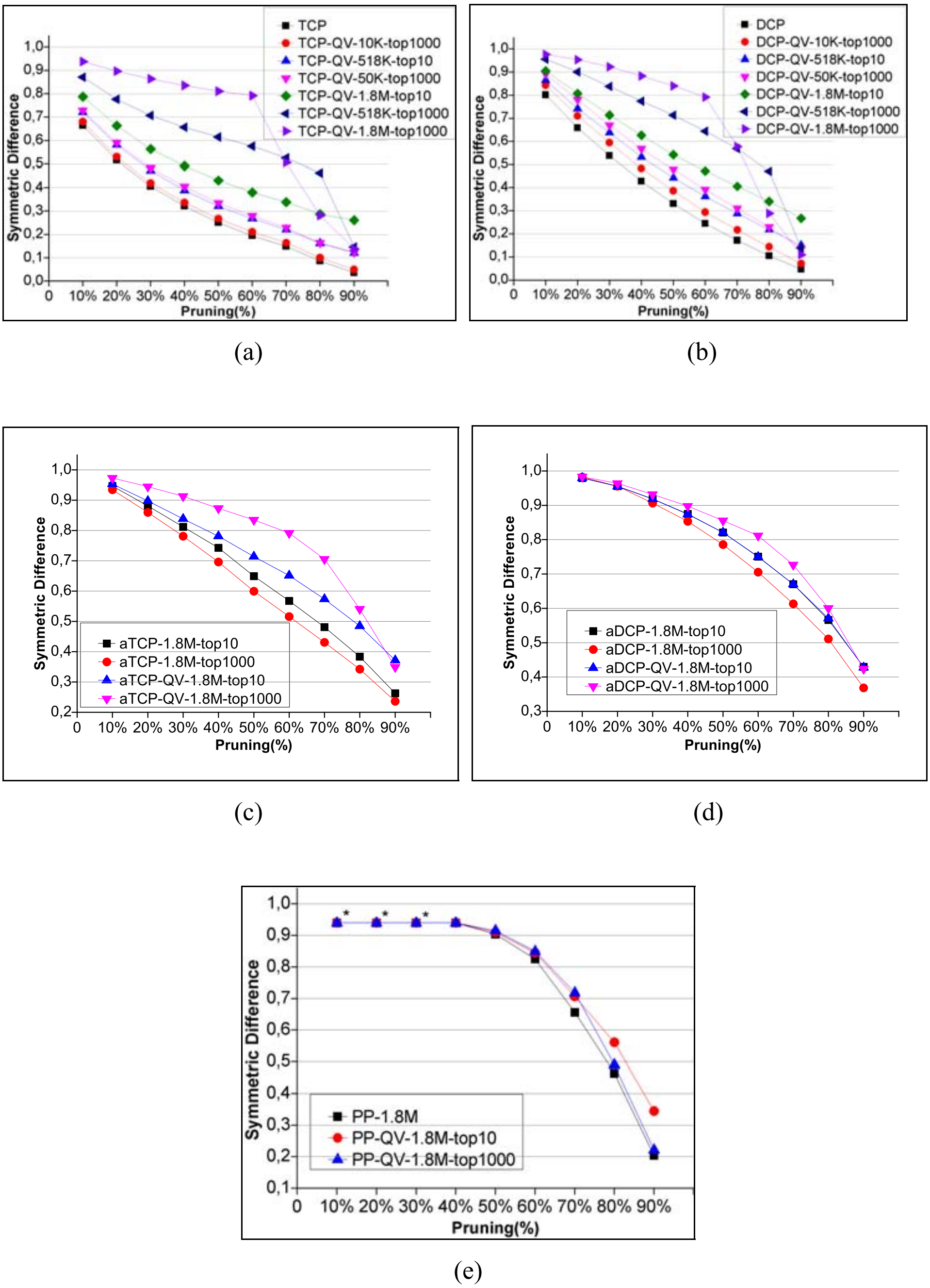

Fig. 12. Retrieval performance of index pruning strategies on the ODP collection using different training sets in conjunctive querying mode: (a) TCP vs. TCP-QV, (b) DCP vs. DCP-QV, (c) aTCP vs. aTCP-QV, (d) aDCP vs. aDCP-QV, and (e) PP vs. PP-QV.

Table II. Avg. symmetric difference scores for top-10 results and disjunctive query processing on ODP collection (relative improvements w.r.t. the baseline algorithms are shown in the column Δ%; all improvements greater than 1% are statistically significant)

%	**TCP**	**DCP**	**aTCP**	**aDCP**	**PP**	**TCP-QV**	**Δ%**	**DCP-QV**	**Δ%**	**aTCP-QV**	**Δ%**	**aDCP-QV**	**Δ%**	**PP-QV**	**Δ%**
10%	0.97	0.94	0.89	0.95	*0.96*	0.97	0%	0.95	1%	0.91	2%	0.95	0%	*0.96*	*0%*
20%	0.91	0.86	0.79	0.90	*0.96*	0.92	1%	0.89	3%	0.84	6%	0.90	0%	*0.96*	*0%*
30%	0.83	0.77	0.69	0.83	*0.96*	0.85	2%	0.82	6%	0.77	12%	0.84	1%	*0.96*	*0%*
40%	0.74	0.68	0.59	0.76	0.96	0.78	5%	0.74	9%	0.70	19%	0.78	3%	0.96	0%
50%	0.64	0.58	0.47	0.67	0.93	0.70	9%	0.67	16%	0.63	34%	0.71	6%	0.93	0%
60%	0.55	0.49	0.38	0.57	0.87	0.62	13%	0.61	24%	0.56	47%	0.63	11%	0.88	1%
70%	0.47	0.40	0.28	0.47	0.75	0.55	17%	0.54	35%	0.50	79%	0.55	17%	0.78	4%
80%	0.35	0.31	0.19	0.34	0.59	0.46	31%	0.47	52%	0.43	126%	0.46	35%	0.66	12%
90%	0.14	0.20	0.09	0.20	0.34	0.40	186%	0.40	100%	0.37	311%	0.38	90%	0.49	44%

In terms of the five baseline algorithms, the findings in this case confirm the earlier observations in [Altingovde et al. 2009a; Carmel et al. 2001; Garcia 2007]. The term-centric adaptation of the access-based approach, aTCP, is the worst among all; and at 50% pruning, the symmetric difference score drops down to 0.47. On the other hand, our document-centric version of the access-based pruning strategy, aDCP, achieves much better performance; it is clearly superior to its term-centric counterpart and provides comparable results to TCP and DCP for most cases.

Nevertheless, we observe that term popularity based pruning strategy, PP, is better than the other four baseline strategies. This is basically due to the fact that PP uses the allocated storage space for the pruned index only for those terms that appear in the previous queries (and have more potential to reappear in the future, as we discuss in Section 6) whereas the other four strategies consider all terms and/or documents. In Section 6, we will also consider hybrid strategies that exploit term popularities.

Next, we evaluate the performance of the strategies with query views, namely TCP-QV, DCP-QV, aTCP-QV, aDCP-QV and PP-QV. A brief glance over Table II reveals that these approaches are far superior to their counterparts that are not augmented with the query views. Remarkably, the order of algorithms is similar in that PP-QV is still the best performer and aTCP-QV is the worst. However, the differences among the absolute effectiveness figures are now much smaller. Indeed, the percentage improvement columns reveal that, query views significantly enhance the performance of the poor strategies at all pruning levels (e.g., gains for aTCP range from 2% to as high as 311%). Even for those strategies that were relatively more successful before, query views provide significant gains, especially at the higher levels of the pruning. For instance, at 90% pruning, the symmetric difference score jumps from 0.34 to 0.49 for PP (a relative

Table III. Avg. symmetric difference scores for top-10 results and conjunctive query processing on ODP collection (relative improvements w.r.t. the baseline algorithms are shown in the column Δ%; all improvements greater than 1% are statistically significant)

%	**TCP**	**DCP**	**aTCP**	**aDCP**	**PP**	**TCP-QV**	**Δ%**	**DCP-QV**	**Δ%**	**aTCP-QV**	**Δ%**	**aDCP-QV**	**Δ%**	**PP-QV**	**Δ%**
10%	0.66	0.80	0.95	0.98	*0.94*	0.79	20%	0.91	14%	0.95	0%	0.98	0%	*0.94*	0%
20%	0.52	0.66	0.88	0.96	*0.94*	0.66	27%	0.81	23%	0.90	2%	0.96	0%	*0.94*	0%
30%	0.41	0.54	0.81	0.92	*0.94*	0.56	37%	0.71	31%	0.84	4%	0.92	0%	*0.94*	0%
40%	0.32	0.43	0.74	0.87	0.94	0.49	53%	0.63	47%	0.78	5%	0.87	0%	0.94	0%
50%	0.25	0.33	0.65	0.82	0.90	0.43	72%	0.54	64%	0.71	9%	0.82	0%	0.91	1%
60%	0.19	0.25	0.57	0.75	0.83	0.38	100%	0.47	88%	0.65	14%	0.75	0%	0.84	1%
70%	0.15	0.17	0.48	0.67	0.66	0.34	127%	0.41	141%	0.57	19%	0.67	0%	0.71	8%
80%	0.09	0.10	0.38	0.57	0.46	0.29	222%	0.34	240%	0.48	26%	0.57	0%	0.56	22%
90%	0.04	0.05	0.26	0.43	0.20	0.26	550%	0.27	440%	0.37	42%	0.43	0%	0.35	75%

increase of 44%), and 0.14 to 0.40 for TCP (186%) using query views. In short, query views significantly improve the baseline strategies, and carry them around 40-50% effectiveness at 90% pruning level, which is a solid success.

An overall comparison of the pruning algorithms: conjunctive mode. In Table III, we provide symmetric difference results in the same setup but for conjunctive query processing mode (on the ODP collection). Interestingly, conjunctive processing is mostly overlooked and has been taken into account in only few works [de Moura et al. 2005; de Moura et al. 2008; Skobeltsyn et al. 2008; Ntoulas and Cho 2007], although it is the default and probably the most crucial processing mode for SEs. Thus, we first analyse the results for the baseline strategies, which has not been discussed in the literature to this extent, before moving to query view based strategies.

Our experiments reveal that for the conjunctive processing mode, TCP is the worst strategy. This is an unsurprising result, of which reasons are discussed in an earlier study [de Moura et al. 2005]. That is, for a conjunctive query including, say, two terms, TCP may have pruned a posting that is at the tail of one term's list and thus reduce the final rank of this posting which is at the top of the other term's list (see Figure 1 in [de Moura et al. 2005]). Furthermore, a TCP-like pruning strategy is also reported to be rather discouraging in ResIn framework [Skobeltsyn et al. 2008]. This is attributed to the observation that the miss-queries are rather discriminative; i.e., return very few results. Recall that our test set also has the same properties as miss-queries, and the average result size is found to be only 398 when top-1000 results per test query are retrieved. Indeed, we created another test set that includes the queries with the highest number of results in our collection and witnessed that TCP's performance can considerably improve.

Nevertheless, in a typical setup with random queries, TCP is the worst performing algorithm for the conjunctive case.

What is more surprising for conjunctive query processing case is the performance of the access-based strategies: aDCP and aTCP outperform TCP and DCP with a wide margin at all pruning levels. This is a new result that has not been reported before in the literature. We think that one reason of this great boost in performance may be the conjunctive processing of the training queries while computing the access counts. In the previous work [Garcia 2007], both training and testing have been conducted in disjunctive mode. We anticipate that the training in conjunctive mode more successfully distinguishes the documents that can also appear in the intersection of terms in other queries. Another remarkable issue is, our document-centric version of the access based strategy, aDCP, significantly outperforms its term-centric adaptation. Indeed, aDCP achieves a similarity of 0.43 to the original results even when the index is reduced to its one tenth (i.e., at 90% pruning level) and outperforms PP. Table III reveals that aDCP and PP are the clear winners for the conjunctive query processing case; the former yields the best result between 90% to 70% pruning, and the latter yields the best results thereafter. The success of aDCP is remarkable, as PP is a very strong competitor. For instance, in ResIn framework, their term+document pruning approach (discussed in more detail in Section 6.1) is reported to yield no improvement over PP when BM25 is used as the ranking function (see Figure 11 in [Skobeltsyn et al. 2008]).

Turning our attention to the query view based strategies, we again report important improvements. This time, the worst performing strategies, TCP and DCP, have most benefited from the query views: TCP-QV and DCP-QV obtain enormous gains especially at the high pruning levels. The gains with aTCP-QV strategy are less emphasized, though reaching up to 42% at 90% pruning. Interestingly, aDCP-QV is found to yield no improvements in comparison to aDCP in this case. This has been also noted for Figure 12(d), where gains are observed only when top-1000 results of the training queries are used. In our detailed analysis for this case, we observed that aDCP and aDCP-QV both select valuable but different postings and the latter has to sacrifice some of these useful postings to open space for the query views; so there happens to be no overall gains.

Finally, PP-QV also improves the performance of PP; for instance the former is relatively 75% better than the latter at 90% pruning level. Still, aDCP-QV (and aDCP) yields the best-performance for the highest pruning levels, namely 80% and 90%, whereas PP-QV yields the best results at all other cases.

6. COMBINING TERM AND DOCUMENT POPULARITIES WITH QUERY VIEW

In Section 5, we show that term popularity based algorithm (PP) performs the best at most of the pruning levels for both disjunctive and conjunctive processing. Another important finding is that especially for the highest pruning levels and conjunctive processing; access-based strategies also serve well and have the potential of outperforming PP. These imply that combining term popularity and document access popularity has the potential of further enhancing the performance.

To justify our intuition, we first conducted a preliminary experiment (on the ODP collection) to show the locality of terms appearing in the queries and top-*k* documents accessed by the queries. For this experiment, we used a test set of 100K queries constructed as described in Section 5. For training, we again employed 1.8M-top10 set. Using these two sets, we first find the number of test queries that include at least one *new term*, i.e., a term that is not encountered in the training set. In our setup, approximately 10% of the test queries involve a new term. Recall that our test query set is essentially composed of singleton queries, i.e., appears only once in the entire query log. Still, 90% of the queries in the test set can be answered by indexing only those terms that are seen in the training log. This explains the success of PP as a pruning algorithm.

Next, we obtain the number of queries that return at least one *new document*, i.e., a document that is never retrieved by the training queries, for top-*k* results. When *k* is set to 10, approximately 13% of the queries retrieve at least one new document. In other words, it is possible to answer all remaining queries by only considering the postings of those documents that are retrieved in the top-10 results of the training queries. Moreover, for top-2 results the new document ratio drops to 4.5%; and for top-1 results only 3% of the queries has a new document. That is, indexing only those documents in the top-10 results of training queries allows correctly identifying the top result for 97% of the queries in the test set.

This experiment indicates that there is a strong potential in exploiting term and document popularities together for improving the pruning strategies. In the literature, term popularity based pruning is combined with some document pruning approaches (e.g., see [Ntoulas and Cho 2007; Skobeltsyn et al. 2008]); however, we are not aware of any work that combine term and document popularities as we propose in this study. In what follows, we first define a general framework to combine the PP algorithm with other strategies, and then evaluate the performance of these algorithms.

***Algorithm** Popularity Pruning (PP)-AAA*

Input: I, μ, popularity[]
1: $L \leftarrow$ sort terms in I the descending order of TermGain(t) = popularity(t)/$|I_t|$
2: $\forall t\ P_t \leftarrow 0$, $AAA_t \leftarrow 0$
3: NumRemainingPostings $\leftarrow$ 0
4: **while** NumRemainingPostings < $|I|$ * (1-μ) **and** L **is not** empty
5: extract term t with the highest gain from L
6: $AAA_t \leftarrow 1$
7: NumRemainingPostings $\leftarrow$ NumRemainingPostings + $|I_{AAA,t}|$
8: reset L as in line (1)
9: **while** NumRemainingPostings < $|I|$ * (1-μ)
10: extract term t with the highest gain from L
11: $P_t \leftarrow 1$
12: NumRemainingPostings $\leftarrow$ NumRemainingPostings + ($|I_t|$- $|I_{AAA,t}|$)
13: **for** each term t in I
14: **if** $P_t = 0$ **and** $AAA_t = 0$
15: remove I_t from I
16: **else if** $P_t = 0$ **and** $AAA_t = 1$
17: remove I_t-$I_{AAA,t}$ from I

Fig. 13. Pseudocode for combining PP approach with the baseline pruning strategies.

6.1 Combined static pruning algorithms with the query views

First, in Figure 13, we outline a general algorithm that combines PP strategy with the other baseline strategies as discussed in this work. We denote a combined algorithm as PP-AAA where AAA can represent either one of the algorithms TCP, DCP, aTCP, or aDCP. In the combined strategy, as in PP strategy, the algorithm proceeds in descending order of the term gain scores. However, in the first pass over the terms (lines 3-6), instead of storing the full lists in the pruned index, the algorithm stores the pruned lists (as generated by the AAA algorithm). If the pruned lists of all of the terms are stored and there is still space (i.e., the size of the pruned index is smaller than the desired size), then the algorithm starts a second pass over term list, again in descending score order. This time, for each term it replaces the pruned list of a term with its full list, until the required file size is reached. When the required pruning level is very high (say, 90%), this strategy allows storing shorter lists (i.e., only pruned lists) and keeping a higher number of terms in the index. But when there is more space, the algorithm can prefer to store more information, i.e., the full lists, of the terms with the highest gain scores, while still keeping the query view of the remaining terms. Using this combination approach, four different algorithms –namely, PP-TCP, PP-DCP, PP-aTCP and PP-aDCP –can be generated, as discussed below.

In the literature, PP-TCP is applied in a slightly different sense: for instance, in the work of Skobeltsyn et al. [2008], the so-called term+document pruning approach keeps a fixed number (denoted as PLL_{max}) of the postings in the index for the terms with the highest gain scores. This has some difficulties in practice: a small PLL_{max} value would practically achieve no pruning whereas a high value may be too crude for smaller lists. Furthermore; PLL_{max} is not correlated to the term gain score: a fixed PLL_{max} value can be pruning half of the postings in, say, the lists of terms with the highest scores while keeping all of the posting for less scoring terms. In our scheme, we first apply a pruning algorithm AAA (at a certain pruning level) to all terms, and then keep the pruned lists for the terms that yield highest gains. The second stage of the algorithm guarantees that, when there is more space available, it is used to favor the highest scoring terms first.

The work of Buttcher and Clarke [2006] is also similar to PP-DCP, in that pruning is only applied for certain terms. However, in their study, they prune the most frequent terms in the index, i.e., those terms with the longest posting lists. In PP-DCP, term popularity is computed from a previous query log and used to compute the term gain score. As another difference, their work assumes that while the pruned lists are kept in the main memory, the full posting lists of the remaining terms are still kept on disk. Here, we assume that if a term's list could not be stored in the pruned index, then it is no more available for querying. Nevertheless, we can state that PP-TCP and PP-DCP are similar to the algorithms discussed in the literature. On the other hand, PP-aTCP and PP-aDCP combine term and document access popularity, and to our knowledge, they are proposed here for the first time in the literature.

Finally, we also propose combining term popularity with the query view augmented strategies. We denote the family of these strategies as PP-AAA-QV (outlined in Figure 14). This is similar to PP-AAA, but in the first pass over the terms, we only attempt to store the postings in the query views. If there is still space in the pruned index, then we store the pruned list of the term, which is obtained using some pruning algorithm AAA-QV. In this algorithm, we never store the full list of a term (unless this term happens to belong to the query views of all documents in its posting list; i.e., $\forall d \in I_t, t \in QV_d$).

In Figures 13 and 14, for the sake of presentation, we assume that the pruned posting lists (i.e., $I_{AAA,t}$ or $I_{AAA\text{-}QV,t}$) are readily available. In an actual implementation, these can be obtained on the fly, depending on the pruning algorithms employed.

Algorithm *Popularity Pruning (PP)-AAA-QV*

```
Input: I, μ, popularity[]
1:  L ← sort terms in I the descending order of TermGain(t)) = popularity(t)/|I_t|
2:  ∀t Q_t ← 0, AAA-QV_t ← 0
3:  NumRemainingPostings ← 0
4:  while NumRemainingPostings < |I| * (1-μ) and L is not empty
5:      extract term t with the highest gain from L
6:      Q_t ← 1
7:      NumRemainingPostings ← NumRemainingPostings +|QV_t|
8:  reset L as in line (1)
9:  while NumRemainingPostings < |I| * (1-μ)
10:     extract term t with the highest gain from L
11:     AAA-QV_t ← 1
12:     NumRemainingPostings ← NumRemainingPostings + (|I_AAA-QV,t|- |QV_t|)
13: for each term t in I
14:     if Q_t = 0 and AAA-QV_t = 0
15:         remove I_t from I
16:     else if Q_t = 1 and AAA-QV_t = 1
17:         remove I_t-I_AAA-QV,t  from I
18:     else if Q_t = 1 and AAA-QV_t = 0
19:         remove I_t-QV_t  from I
```

Fig. 14. Pseudocode for combining PP approach with the QV based pruning strategies.

6.2 Experimental Evaluation

Performance of the combined algorithms: disjunctive mode. In Table IV, we present the performance of combined pruning algorithms for disjunctive query processing on the ODP collection. The effectiveness figures for PP and PP-QV are copied from Table II to ease the comparisons. For the experiments, we again used 1.8M-top10 training set, as in Section 5.

As mentioned in the above, the combined algorithms employ a pruning algorithm, denoted as AAA or AAA-QV, at a certain pruning level. For all cases, we experimented with 10%, 30% and 50% levels; and set the pruning level as 50% as it yields the best performance for the combined algorithms at high pruning levels. Note that, the total size of the pruned posting lists (generated by AAA or AAA-QV) for the terms that appear in the training set corresponds to at most 40% of the full index. So, we report values at pruning levels equal to or greater than 60%. We believe that, these high pruning levels are the cases that demand for improvements utmost; as at the moderate pruning levels it is actually possible to obtain more than 90% effectiveness.

We can summarize the findings drawn from Table IV as follows: First of all, when we compare the baseline algorithms, we see that PP-aTCP is worse than PP, whereas the remaining algorithms, PP-TCP, PP-DCP and PP-aDCP can outperform PP at high pruning levels. For instance, all latter strategies are significantly better (based on the

Table IV. Avg. symmetric difference scores for top-10 results and disjunctive query processing on the ODP collection (all differences between PP-AAA and PP-AAA-QV algorithms are statistically significant; relative improvements are shown in parentheses)

%	PP	PP-QV	PP-TCP	PP-TCP-QV	PP-DCP	PP-DCP-QV	PP-aTCP	PP-aTCP-QV	PP-aDCP	PP-aDCP-QV
60%	0.87	0.88 (1%)	0.73	0.79 (8%)	0.63	0.73 (16%)	0.47	0.65 (38%)	0.70	0.74 (6%)
70%	0.75	0.78 (4%)	0.73	0.79 (8%)	0.63	0.73 (16%)	0.47	0.65 (38%)	0.69	0.73 (6%)
80%	0.59	0.66 (12%)	0.67	0.72 (7%)	0.59	0.68 (15%)	0.44	0.61 (39%)	0.60	0.66 (10%)
90%	0.34	0.49 (44%)	0.47	0.54 (15%)	0.41	0.52 (27%)	0.32	0.49 (53%)	0.40	0.50 (25%)

ANOVA results) than PP at 90% pruning. For all strategies, embedding query views cause significant improvements. Among the algorithms with query views, PP-TCP-QV again yields the best results at pruning levels between 70% and 90%.

In short, for the disjunctive case, if the query views are not employed, PP-TCP provides the best performance at high pruning levels, namely at 80-90% pruning, while PP-DCP and PP-aDCP can also outperform PP at these level. When query views based algorithms are used, the relative order of the algorithms remain the same; but the gaps between their performances are reduced as query views provide the highest and lowest improvements for PP and PP-TCP, respectively. These results may be expected, as access-based strategies are more competitive especially for the conjunctive case. In what follows, we analyse the latter case and show that it is possible to obtain higher effectiveness by combining term and document access popularity.

Performance of the combined algorithms: conjunctive mode. In Table V, we compare the effectiveness of the combined pruning strategies for conjunctive query processing on the ODP collection. A comparison among the baseline algorithms reveals that PP-TCP and PP-DCP are both worse than PP, whereas PP-aTCP (at 80% and 90% levels) and PP-aDCP (at 70%-90%) outperform PP (all differences are significant based on ANOVA). For instance, PP achieves a symmetric difference score of only 0.20 at 90% pruning level, whereas both PP-aTCP and PP-aDCP yield 0.32, indicating a relative improvement of 60%. As another important result, we see that all algorithms are considerably improved by using the query views. In the conjunctive case, the approaches that most and least benefit from the query views are PP-DCP-QV and PP-aDCP-QV, respectively. The gain for the PP-DCP-QV is almost 127% at the 90% pruning level. Even PP-aDCP-QV experiences a relative improvement of 22% at that level.

Table V. Avg. symmetric difference scores for top-10 results and conjunctive query processing on the ODP (all differences between PP-AAA and PP-AAA-QV algorithms – except the cases marked with a (*), are stat. significant; improvements are in parentheses)

%	PP	PP-QV	PP-TCP	PP-TCP-QV	PP-DCP	PP-DCP-QV	PP-aTCP	PP-aTCP-QV	PP-aDCP	PP-aDCP-QV
60%	0.83	0.84 (1%)	0.28	0.46 (64%)	0.30	0.51 (70%)	0.62	0.69 (11%)	0.79	0.79*
70%	0.66	0.71 (8%)	0.28	0.46 (64%)	0.30	0.51 (70%)	0.62	0.69 (11%)	0.77	0.77*
80%	0.46	0.56 (22%)	0.25	0.42 (68%)	0.26	0.47 (81%)	0.55	0.62 (13%)	0.62	0.64 (3%)
90%	0.20	0.35 (75%)	0.14	0.31 (121%)	0.15	0.34 (127%)	0.32	0.40 (25%)	0.32	0.39 (22%)

An overall consideration of the results in Table V demonstrates that, especially at the very high pruning levels, between 70%-90%, combining term and document popularities pays off, and indeed augmenting these strategies with query views further improves the effectiveness. At 90% pruning level, PP-aDCP-QV yields an effectiveness score of 0.39, almost twice as better than PP, which is a very competitive baseline as shown in the recent works. For smaller pruning levels (i.e., less than 70%), PP is still the best performer and there is not much gain in coupling it with the query view or other pruning strategies.

Performance of the combined algorithms: different test sets. We conducted additional experiments involving a set of 100K random queries that is constructed as described in Section 5.1. Due to time and resource limitations, we experimented with only those cases reported in Table V, i.e., the most promising results for conjunctive processing. We observed almost the same effectiveness figures and trends in every aspect; i.e., findings on the test-1000 set are also confirmed by the results obtained for the large test set.

Moreover, we also investigated whether test sets that correspond to different time periods (i.e., especially to time periods that have larger temporal distance to submission times of training queries) yield different results; and again, observed no meaningful difference. Please see Appendix 2 for the details.

Performance of the combined algorithms: ClueWeb09-B collection. Given that the combined algorithms with query views constitute the best-performing family of static pruning strategies proposed in this study, we further investigate their performance on a larger collection, namely, ClueWeb09-B. In particular, we repeated all experiments reported in Tables IV and V. As in the ODP collection case, we again set pruning level to 50% for AAA and AAA-QV algorithms used in the combined experiments. A minor distinction from the previous case is in terms of the training set. While we employ the training set of 1.8M queries with top-10 results for the evaluation on the ODP collection,

Table VI. Avg. symmetric difference scores for top-10 results and disjunctive query processing on ClueWeb09-B (relative improvements by QVs are shown in parentheses)

%	PP	PP-QV	PP-TCP	PP-TCP-QV	PP-DCP	PP-DCP-QV	PP-aTCP	PP-aTCP-QV	PP-aDCP	PP-aDCP-QV
60%	0.76	0.79 (4%)	0.76	0.80 (5%)	0.68	0.68 (0%)	0.62	0.69 (11%)	0.75	0.76 (1%)
70%	0.63	0.69 (10%)	0.74	0.78 (5%)	0.66	0.66 (0%)	0.61	0.68 (11%)	0.73	0.74 (1%)
80%	0.46	0.56 (22%)	0.62	0.67 (8%)	0.54	0.56 (4%)	0.52	0.60 (15%)	0.60	0.64 (7%)
90%	0.23	0.40 (74%)	0.39	0.49 (26%)	0.32	0.41 (28%)	0.33	0.45 (36%)	0.35	0.46 (31%)

Table VII. Avg. symmetric difference scores for top-10 results and conjunctive query processing on ClueWeb09-B (relative improvements by QVs are shown in parentheses)

%	PP	PP-QV	PP-TCP	PP-TCP-QV	PP-DCP	PP-DCP-QV	PP-aTCP	PP-aTCP-QV	PP-aDCP	PP-aDCP-QV
60%	0.73	0.76 (4%)	0.61	0.67 (10%)	0.54	0.54 (0%)	0.63	0.69 (10%)	0.76	0.76 (0%)
70%	0.58	0.65 (12%)	0.59	0.65 (10%)	0.52	0.52 (0%)	0.62	0.67 (8%)	0.73	0.73 (0%)
80%	0.39	0.50 (28%)	0.48	0.55 (15%)	0.41	0.42 (2%)	0.51	0.57 (12%)	0.56	0.61 (9%)
90%	0.16	0.32 (100%)	0.24	0.38 (58%)	0.21	0.28 (33%)	0.28	0.39 (39%)	0.28	0.40 (43%)

here we prefer to use top-100 results for the same query set. This is because ClueWeb09-B collection is much larger than the ODP and 1.8M-top10 training set yields a very small query view with respect to the original index size.

In Tables VI and VII, we present the performance of combined pruning algorithms for disjunctive and conjunctive query processing semantics, respectively. The tables reveal that, the absolute scores are lower than those obtained for the ODP collection and there are slight differences among the relative order of algorithms. Most remarkably, the performance of PP is inferior to PP-AAA algorithms for pruning levels greater than 60% for both disjunctive and conjunctive processing. This might be due to storing full lists for popular terms, which might also include several redundant postings in case of a very large collection, whereas all PP-AAA algorithms first apply a pruning of 50% using the corresponding AAA algorithm. Nevertheless, a glance on these tables shows that using query views still improves all algorithms in almost all cases. Similar to previous findings in the ODP case, PP-TCP-QV and PP-aDCP-QV yield the best results for disjunctive and conjunctive semantics, respectively. We also provide results that exhibit similar trends using a standard evaluation metric, namely MAP, in Appendix 3.

Additionally, for 50 queries with relevance judgments (used in the TREC 2009 Web Track [Clark et al. 2010]), we obtained stat mean nDCG (over top-1000 results) as well as traditional MAP (over top-10 results) and P@10 scores for the cases reported in the Tables VI and VII. We verified that our key finding still holds, i.e., combining PP with other strategies and then with query views significantly improve the performance especially at high pruning levels (see Appendix 4 for the details).

Table VIII. Average percentage of data fetched from disk during query processing

%	PP	PP-QV	PP-TCP	PP-TCP-QV	PP-DCP	PP-DCP-QV	PP-aTCP	PP-aTCP-QV	PP-aDCP	PP-aDCP-QV
60%	88.5%	88.5%	63.9%	65.8%	46.3%	45.7%	61.9%	63.8%	63.8%	63.1%
70%	77.3%	77.3%	63.3%	65.1%	45.8%	45.3%	61.3%	63.1%	63.1%	62.2%
80%	63.3%	63.3%	57.0%	58.6%	41.1%	41.1%	55.3%	56.8%	56.8%	54.7%
90%	44.1%	44.1%	40.7%	42.0%	29.2%	30.3%	39.4%	40.8%	41.0%	39.5%

We also investigate the query processing efficiency on the pruned index files produced by the combined pruning algorithms. As disk access costs usually dominate the query processing cost, for each pruning strategy, we provide the average percentage of data transferred from disk (in comparison to the full index) during query processing on the ClueWeb09-B dataset. Note that, all data transfer costs are based on the compressed posting list sizes, i.e.; both document-id gaps and term frequency values in the postings are compressed using Elias- -γ encoding scheme. We report the results in Table VIII.

Table VIII reveals several interesting aspect. First of all, the results show that while static index pruning significantly reduces the storage space for index files, its benefits in terms of disk transfer costs are more conservative as the most popular query terms can be pruned only moderately. For instance, PP algorithm (by definition) stores the full lists of selected terms; thus, in terms of data transfer from disk, it yields the minimum gain: at 60% pruning level, the amount of data transferred is 88.5% of the data that would be transferred from the full index while processing test-1000 set. All PP-AAA algorithms incur disk transfer costs less than PP, usually ranging from 60% to 40% of those of the full index for the pruning levels from 60% to 90%, respectively. We see that PP-DCP has remarkably lower disk costs than other PP-AAA approaches, which implies that it has pruned the most popular query terms harshly. This also explains the inferior performance of DCP and DCP-QV algorithms especially in Table VII. The other three approaches, PP-TCP, PP-aTCP and PP-aDCP, incur similar data transfer costs that are all lower than that of PP at a given pruning level. From our perspective, a positive finding is that PP-aDCP, which is superior to other variants and PP in terms of effectiveness, also incurs comparable costs to PP-TCP and PP-aTCP. This means that PP-aDCP embodies a very effective decision mechanism for choosing the most valuable postings and eliminating all others.

Another crucial finding is that while augmenting PP-AAA algorithms with query views significantly improve their effectiveness, the impact on the data transfer costs is negligible. For all results in Table VIII, we computed the relative difference in the amount of transferred data between corresponding PP-AAA and PP-AAA-QV algorithms

Table IX. Average percentage of postings accessed during query processing.

%	PP	PP-QV	PP-TCP	PP-TCP-QV	PP-DCP	PP-DCP-QV	PP-aTCP	PP-aTCP-QV	PP-aDCP	PP-aDCP-QV
60%	86.8%	86.7%	49.2%	51.0%	27.4%	26.9%	49.2%	51.0%	48.9%	49.2%
70%	74.8%	74.8%	48.6%	50.4%	27.1%	26.6%	48.6%	50.4%	48.1%	48.4%
80%	60.6%	60.5%	43.4%	44.9%	24.4%	24.2%	43.4%	44.9%	42.1%	42.1%
90%	42.3%	42.3%	30.3%	31.1%	17.4%	17.6%	30.3%	31.1%	29.3%	29.5%

and observed that the change is rather negligible (i.e., between -3.5% to +3.5%). That is, query views increased the amount of data transferred in some cases (e.g., compare PP-TCP and PP-TCP-QV columns in Table VIII) whereas they reduced the transfer costs in some other cases (e.g., for PP-aDCP-QV). Remarkably, the best performing algorithm for more common conjunctive query semantics, namely, PP-aDCP-QV has even lower disk transfer costs than PP-aDCP.

In a separate experiment, we also investigate the average percentage of the number of postings accessed during query processing to represent in-memory query processing costs. Note that, we don't take into account the query processing semantics or other dynamic pruning techniques and assumed that all postings of the query terms are accessed. While the actual number of scored postings may vary due to these latter issues, we believe that number of postings is still a reasonable approximation for comparing the efficiency of corresponding PP-AAA and PP-AAA-QV algorithms, as both would benefit from other optimizations (such as dynamic pruning) in similar ways, in practice. This experiment again shows that possible increments in the number of postings that are due to incorporating query views into PP-AAA pruning strategies are minor, e.g., PP-aDCP-QV causes a relative increase of at most 0.8% in the number of accessed postings with respect to PP-aDCP (at 90% pruning level). Results of this experiment are reported in Table IX.

7. CONCLUSIONS AND FUTURE WORK

In this study, we first propose query view based strategies for static pruning to improve the top-ranked result quality. We incorporate query views into a number of strategies that exist in the literature and/or adapted by us. Additionally, we propose new pruning strategies that combine the notions of term and document access popularity, and then couple them again with the query views. We summarize our key findings as follows:

- Using query views significantly improves almost all of the baseline pruning strategies for both disjunctive and conjunctive processing. For most cases, gains can be obtained by using a training set constructed by using only top-10 results for the queries in the set. Such a training set can be simply formed during actual query processing of a SE.

- Regarding the training query set sizes, we draw the following conclusion to guide practitioners. Suppose that a certain level of performance is observed on some reference collection *C1* using query view based pruning algorithms and a training set *Q1*. To obtain the same level of performance on another collection *C2*, it is necessary to use a training query set *Q2* that would yield a query view for *C2* whose ratio to *C2*'s full index is at least as large as the ratio of *C1*'s query view size to its own full index size. Note that, in such a case, both training query sets to be employed for creating query views on collections *C1* and *C2* should better use the same number of top-*k* results.
- Combining PP with the other strategies further improve performance. In terms of the combination of PP with the other four baseline strategies (i.e., PP-AAA algorithms), we see that different combinations can outperform PP at different cases. For disjunctive query processing, PP-TCP and PP-aDCP outperform PP at high pruning levels. For conjunctive case, combination of term and document access popularity pays off: PP-aTCP and PP-aDCP are superior to PP for the highest pruning levels.
- Query views further enhance the performance of the combined approaches. For disjunctive processing, PP-QV performs very well, whereas PP-TCP-QV and PP-aDCP-QV yield comparable or better performance. For conjunctive case, PP-aDCP-QV again produces the highest effectiveness figures.

Our work presented in this paper covers several aspects of static index pruning. Still, there are a number of issues that cannot be considered within the scope of a single work. First, our pruning strategies aim to improve the overlap in top-ranked results obtained from the pruned indexes and the full index to the greatest extent possible. An alternative goal in this setup could be providing exactly the same top-ranked results for as many as queries possible, as in [Ntoulas and Cho 2007], which is not explored in this study. Second, the impacts of the alternative ranking functions that exploit term proximity models or query-independent features are not taken into account. These issues are left as future research directions.

Finally, static pruning techniques in the literature usually overlook the dynamicity of the underlying index. However, in practice, Web pages change frequently; enforcing the update of the index that is built on top of them. In the literature, three different methods (and their variants) for index update are discussed, namely, re-build, re-merge and in-place update (see [Zobel and Moffat 2006] for a survey). The simplest strategy is periodically re-building the index. In this case, the pruned index (or indexes) can be obtained at the same time. Re-merge technique grows one or more delta index files to be merged to the main index, whereas the in-place update technique leaves some free space at the end of each list for the new postings. For these latter approaches, it is not clear how to update the pruned index or when to regenerate it. We also leave investigating the answers for these questions as a future work.

ACKNOWLEDGEMENTS

We thank Cigdem Gündüz-Demir and Ben Carterette for the valuable discussions on the statistical tests. We also thank Isik Ayranci, Sadiye Kaptanoglu, and Levent Koc for their contributions for the implementation. We are grateful to Duygu Atilgan for her valuable feedback on earlier drafts of this study.

REFERENCES

ALTINGOVDE, I. S., DEMIR, E., CAN, F., AND ULUSOY, Ö. 2008. Incremental cluster-based retrieval using compressed cluster-skipping inverted files. *ACM Transactions on Information Systems*, 26, 3, 1–36.

ALTINGOVDE, I. S., OZCAN, R., AND ULUSOY, Ö. 2009a. A practitioner's guide for static index pruning. In *Proceedings of the 31st European Conference on IR Research (ECIR)*, pp. 675–679.

ALTINGOVDE, I. S., OZCAN, R., AND ULUSOY, Ö. 2009b. Exploiting query views for static index pruning in Web search engines. In *Proceedings of the 18th ACM International Conference on Information and Knowledge Management (CIKM)*, pp. 1951–1954.

ANH, N.V., KRETSER, O. DE, AND MOFFAT, A. 2001. Vector-space ranking with effective early termination. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, pp. 35–42

ANTONELLIS, I. AND GARCIA-MOLINA, H. AND KARIM, J. 2009. Tagging with queries: how and why? In *Proceedings of the Second International Conference on Web Search and Web Data Mining (Late breaking-results)*.

BAEZA-YATES, R., GIONIS, A., JUNQUEIRA, F., MURDOCK, V., PLACHOURAS, V., AND SILVESTRI, F. 2007. The impact of caching on search engines. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, New York, NY, pp. 183–190.

BLANCO, R. 2008. Index Compression for Information Retrieval Systems. Doctoral Thesis, University of A Coruña.

BLANCO, R. AND BARREIRO, A. 2007a. Boosting static pruning of inverted files. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, New York, NY, pp. 777–778.

BLANCO, R. AND BARREIRO. A. 2007b. Static pruning of terms in inverted files. In *Proceedings of the 29th European Conference on IR Research (ECIR)*, pp. 64–75.

BÜTTCHER, S. AND CLARKE, C. L. 2006. A document-centric approach to static index pruning in text retrieval systems. In *Proceedings of the 15th ACM International Conference on Information and Knowledge Management (CIKM)*, pp. 182–189.

CAMBAZOGLU B. B., PLACHOURAS, V. AND BAEZA-YATES, R. 2009. Quantifying performance and quality gains in distributed web search engines. In *Proceedings of the 32th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, New York, NY, pp. 411–418.

CARMEL, D., COHEN, D., FAGIN, R., FARCHI, E., HERSCOVICI, M., MAAREK, Y. S., AND SOFFER, A. 2001. Static index pruning for information retrieval systems. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, pp. 43–50.

CASTELLANOS, M. 2003. Hotminer: Discovering hot topics from dirty text. In M. W. Berry, editor, *Survey of Text Mining*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

CLARKE, C. L., CRASWELL, N., AND SOBOROFF, I. 2010. Overview of the TREC 2009 Web track. In *Proceedings of the 18th Text Retrieval Conference*, E. M. Voorhees and L. P. Buckland, Eds. NIST, Gaithersburg, MD, USA.

DE MOURA, E. S., DOS SANTOS, C. F., FERNANDES, D. R., SILVA, A. S., CALADO, P. AND NASCIMENTO, M. A. 2005. Improving Web search efficiency via a locality based static pruning method. In *Proceedings of the 14th International WWW Conference*, pp. 235–244.

DE MOURA, E. S., SANTOS, C. F., ARAUJO, B. D., SILVA, A. S., CALADO, P., AND NASCIMENTO, M. A. 2008. Locality-Based pruning methods for web search. *ACM Transactions on Information Systems*, 26, 2, 1–28.

GARCIA, S. 2007. Search Engine Optimization Using Past Queries. Doctoral Thesis, RMIT University.

GARCIA, S. 2009. RMIT University at TREC 2009: Web Track. In *Proceedings of the Eighteenth Text Retrieval Conference (TREC).*

GARCIA, S., WILLIAMS, H. E., AND CANNANE, A. 2004. Access-ordered indexes. In *Proceedings of the Twenty-Seventh Australasian Computer Science Conference (ACSC)*, pp. 7–14.

GARCIA, S. AND TURPIN, A. 2006. Efficient Query Evaluation Through Access Reordering. In *Proceedings of the 3rd Asia Information Retrieval Symposium (AIRS),* pp. 106–118.

JANSEN, B.J., SPINK, A. 2000. Methodological approach in discovering user search patterns through Web log analysis. *ACM SIGIR Forum 32*, 5–17.

KAPTEIN, R., KOOLEN, M., KAMPS, J. 2009. Result Diversity and Entity Ranking Experiments*:* Anchors*,* Links*,* Text and Wikipedia. In *Proceedings of the 18th Text Retrieval Conference*, E. M. Voorhees and L. P. Buckland, Eds. NIST, Gaithersburg, MD, USA.

MOFFAT, A. AND ZOBEL, J. 1996. Self-indexing inverted files for fast text retrieval. *ACM Transactions on Information Systems*, 14, 4, 349–379.

NTOULAS, A. AND CHO, J. 2007. Pruning policies for two-tiered inverted index with correctness guarantee. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, New York, NY, pp. 191–198.

OZCAN, R., ALTINGOVDE, I. S., AND ULUSOY, Ö. 2011. Cost-Aware Strategies for Query Result Caching in Web Search Engines. *ACM Transactions on Web*, to appear.

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.

POBLETE, B. AND BAEZA-YATES, R. 2008. Query-sets: using implicit feedback and query patterns to organize web documents. In *Proceedings of the 17th International WWW Conference*, pp. 41–50.

PUPPIN, D., SILVESTRI, F., AND LAFORENZA, D. 2006. Query-driven document partitioning and collection selection. In *Proceedings of the 1st International Conference on Scalable Information Systems*, p. 34.

PUPPIN, D., PEREGO, R., SILVESTRI, F., AND BAEZA-YATES, R. 2010. Tuning the Capacity of Search Engines: Load-driven Routing and Incremental Caching to Reduce and Balance the Load. *ACM Transactions on Information Systems*, 28, 2.

SKOBELTSYN, G., JUNQUEIRA, F., PLACHOURAS, V., AND BAEZA-YATES, R. 2008. ResIn: a combination of results caching and index pruning for high-performance web search engines. In *Proceedings of the 31st*

Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. ACM, New York, NY, pp. 131–138.

TONELLOTTO, N., MACDONALD, C., AND OUNIS, I. 2010. Efficient dynamic pruning with proximity support. In *Proceedings of LSDS-IR Workshop at SIGIR,* pp. 31–35.

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

APPENDIX 1. EXPERIMENTAL RESULTS FOR DIFFERENT TRAINING SETS

For the astute reader, here we provide detailed results corresponding to Figures 11 and 12 for all six training sets. Note that, each table is in the flavor of Tables II and III given in Section 5.2; i.e., the first six tables (Tables A1.1 to A1.6) provide the average symmetric difference scores for the six training sets given in Table I and disjunctive query processing semantics; and the second six tables (Tables A1.7 to A1.12) provide results for conjunctive query processing semantics. For the smallest two training sets (10K and 50K), we haven't obtained the results with PP and PP-QV, as they were not promising.

Finally note that, for some cases below, it is seen that a particular algorithm (especially for our query view based versions) performs better than the 1.8M-top10 case that we chose to present in detail in the previous sections. For instance, TCP-QV performs better for 1.8M-top1000 training set at low pruning levels; a point which is more easily understandable from the Figures 11(a) and 11(b). Nevertheless, we preferred to present 1.8M top 10 results, as it is the largest query set and it is more practical to obtain/store top-10 results in a real system (as they are already generated during the typical service of a search engine).

Table A1.1. Avg. symmetric difference scores for top-10 results and disjunctive query processing using *1.8M-top10 training set*. (This corresponds to Table II above and repeated here to ease comparisons).

%	TCP	DCP	aTCP	aDCP	PP	TCP-QV	Δ%	DCP-QV	Δ%	aTCP-QV	Δ%	aDCP-QV	Δ%	PP-QV	Δ%
10%	0.97	0.94	0.89	0.95	*0.96*	0.97	0%	0.95	1%	0.91	2%	0.95	0%	*0.96*	*0%*
20%	0.91	0.86	0.79	0.90	*0.96*	0.92	1%	0.89	3%	0.84	6%	0.90	0%	*0.96*	*0%*
30%	0.83	0.77	0.69	0.83	*0.96*	0.85	2%	0.82	6%	0.77	12%	0.84	1%	*0.96*	*0%*
40%	0.74	0.68	0.59	0.76	0.96	0.78	5%	0.74	9%	0.70	19%	0.78	3%	0.96	0%
50%	0.64	0.58	0.47	0.67	0.93	0.70	9%	0.67	16%	0.63	34%	0.71	6%	0.93	0%
60%	0.55	0.49	0.38	0.57	0.87	0.62	13%	0.61	24%	0.56	47%	0.63	11%	0.88	1%
70%	0.47	0.40	0.28	0.47	0.75	0.55	17%	0.54	35%	0.50	79%	0.55	17%	0.78	4%
80%	0.35	0.31	0.19	0.34	0.59	0.46	31%	0.47	52%	0.43	126%	0.46	35%	0.66	12%
90%	0.14	0.20	0.09	0.20	0.34	0.40	186%	0.40	100%	0.37	311%	0.38	90%	0.49	44%

Table A1.2. Avg. symmetric difference scores for top-10 results and disjunctive query processing using *1.8M-top1000 training set*.

%	**TCP**	**DCP**	**aTCP**	**aDCP**	**PP**	**TCP-QV**	**Δ%**	**DCP-QV**	**Δ%**	**aTCP-QV**	**Δ%**	**aDCP-QV**	**Δ%**	**PP-QV**	**Δ%**
10%	0.97	0.94	0.84	0.94	*0.96*	0.98	1%	0.98	4%	0.93	11%	0.96	2%	*0.96*	*0%*
20%	0.91	0.86	0.68	0.87	*0.96*	0.95	4%	0.95	10%	0.88	29%	0.91	5%	*0.96*	*0%*
30%	0.83	0.77	0.54	0.77	*0.96*	0.91	10%	0.93	21%	0.84	56%	0.86	12%	*0.96*	*0%*
40%	0.74	0.68	0.42	0.66	0.96	0.86	16%	0.89	31%	0.8	90%	0.81	23%	0.96	0%
50%	0.64	0.58	0.31	0.54	0.93	0.82	28%	0.84	45%	0.76	145%	0.77	43%	0.94	1%
60%	0.55	0.49	0.22	0.41	0.87	0.79	44%	0.79	61%	0.74	236%	0.74	80%	0.87	-1%
70%	0.47	0.40	0.14	0.3	0.75	0.71	51%	0.66	65%	0.62	343%	0.66	120%	0.74	-3%
80%	0.35	0.31	0.08	0.19	0.59	0.62	77%	0.48	55%	0.32	300%	0.39	105%	0.58	0%
90%	0.14	0.20	0.03	0.08	0.34	0.44	214%	0.31	55%	0.12	300%	0.16	100%	0.34	1%

Table A1.3. Avg. symmetric difference scores for top-10 results and disjunctive query processing using *518K-top10 training set*.

%	**TCP**	**DCP**	**aTCP**	**aDCP**	**PP**	**TCP-QV**	**Δ%**	**DCP-QV**	**Δ%**	**aTCP-QV**	**Δ%**	**aDCP-QV**	**Δ%**	**PP-QV**	**Δ%**
10%	0.97	0.94	0.88	0.92	*0.90*	0.97	0%	0.94	0%	0.88	0%	0.92	0%	*0.90*	*0%*
20%	0.91	0.86	0.78	0.86	*0.90*	0.92	1%	0.88	2%	0.79	1%	0.86	0%	*0.90*	*0%*
30%	0.83	0.77	0.67	0.8	*0.90*	0.84	1%	0.79	3%	0.7	4%	0.8	0%	*0.90*	*0%*
40%	0.74	0.68	0.57	0.74	0.90	0.76	3%	0.71	4%	0.62	9%	0.74	0%	0.90	0%
50%	0.64	0.58	0.46	0.65	0.89	0.67	5%	0.63	9%	0.54	17%	0.66	2%	0.89	0%
60%	0.55	0.49	0.37	0.56	0.83	0.59	7%	0.56	14%	0.46	24%	0.58	4%	0.84	1%
70%	0.47	0.4	0.27	0.45	0.71	0.51	9%	0.48	20%	0.39	44%	0.48	7%	0.73	3%
80%	0.35	0.31	0.18	0.33	0.56	0.40	14%	0.40	29%	0.32	78%	0.38	15%	0.59	5%
90%	0.14	0.20	0.09	0.19	0.32	0.29	107%	0.32	60%	0.25	178%	0.28	47%	0.40	25%

Table A1.4. Avg. symmetric difference scores for top-10 results and disjunctive query processing using *518K-top1000 training set*.

%	**TCP**	**DCP**	**aTCP**	**aDCP**	**PP**	**TCP-QV**	**Δ%**	**DCP-QV**	**Δ%**	**aTCP-QV**	**Δ%**	**aDCP-QV**	**Δ%**	**PP-QV**	**Δ%**
10%	0.97	0.94	0.84	0.94	*0.90*	0.97	0%	0.96	2%	0.9	7%	0.95	1%	*0.90*	*0%*
20%	0.91	0.86	0.69	0.87	*0.90*	0.93	2%	0.92	7%	0.81	17%	0.88	1%	*0.90*	*0%*
30%	0.83	0.77	0.54	0.77	*0.90*	0.87	5%	0.87	13%	0.74	37%	0.8	4%	*0.90*	*0%*
40%	0.74	0.68	0.42	0.66	0.90	0.8	8%	0.82	21%	0.67	60%	0.72	9%	0.90	0%
50%	0.64	0.58	0.31	0.54	0.89	0.74	16%	0.77	33%	0.6	94%	0.64	19%	0.89	0%
60%	0.55	0.49	0.22	0.41	0.83	0.68	24%	0.71	45%	0.55	150%	0.56	37%	0.84	1%
70%	0.47	0.4	0.14	0.29	0.71	0.59	26%	0.62	55%	0.5	257%	0.5	72%	0.72	1%
80%	0.35	0.31	0.08	0.19	0.56	0.45	29%	0.48	55%	0.46	475%	0.46	142%	0.51	-9%
90%	0.14	0.20	0.03	0.08	0.32	0.39	179%	0.30	50%	0.18	500%	0.21	163%	0.33	3%

Table A1.5. Avg. symmetric difference scores for top-10 results and disjunctive query processing using *50K-top1000 training set*.

%	TCP	DCP	aTCP	aDCP	TCP-QV	Δ%	DCP-QV	Δ%	aTCP-QV	Δ%	aDCP-QV	Δ%
10%	0.97	0.94	0.83	0.92	0.97	0%	0.95	1%	0.84	1%	0.92	0%
20%	0.91	0.86	0.66	0.84	0.92	1%	0.88	2%	0.7	6%	0.84	0%
30%	0.83	0.77	0.53	0.74	0.85	2%	0.81	5%	0.57	8%	0.74	0%
40%	0.74	0.68	0.42	0.62	0.76	3%	0.72	6%	0.47	12%	0.62	0%
50%	0.64	0.58	0.3	0.51	0.67	5%	0.64	10%	0.37	23%	0.51	0%
60%	0.55	0.49	0.22	0.4	0.58	5%	0.56	14%	0.29	32%	0.41	2%
70%	0.47	0.4	0.14	0.29	0.5	6%	0.48	20%	0.22	57%	0.3	3%
80%	0.35	0.31	0.08	0.17	0.37	6%	0.38	23%	0.16	100%	0.20	18%
90%	0.14	0.20	0.03	0.07	0.24	71%	0.26	30%	0.11	267%	0.11	57%

Table A1.6. Avg. symmetric difference scores for top-10 results and disjunctive query processing using *10K-top1000 training set*.

%	TCP	DCP	aTCP	aDCP	TCP-QV	Δ%	DCP-QV	Δ%	aTCP-QV	Δ%	aDCP-QV	Δ%
10%	0.97	0.94	0.8	0.87	0.97	0%	0.94	0%	0.8	0%	0.87	0%
20%	0.91	0.86	0.64	0.77	0.91	0%	0.87	1%	0.65	2%	0.77	0%
30%	0.83	0.77	0.51	0.68	0.83	0%	0.78	1%	0.52	2%	0.68	0%
40%	0.74	0.68	0.4	0.6	0.74	0%	0.69	1%	0.41	2%	0.6	0%
50%	0.64	0.58	0.29	0.47	0.65	2%	0.6	3%	0.31	7%	0.47	0%
60%	0.55	0.49	0.2	0.37	0.56	2%	0.51	4%	0.23	15%	0.37	0%
70%	0.47	0.4	0.13	0.26	0.48	2%	0.43	7%	0.16	23%	0.26	0%
80%	0.35	0.31	0.07	0.16	0.36	3%	0.33	6%	0.10	43%	0.16	0%
90%	0.14	0.20	0.03	0.07	0.17	21%	0.23	15%	0.05	67%	0.07	0%

Table A1.7. Avg. symmetric difference scores for top-10 results and conjunctive query processing using *1.8M-top10* training set. (This corresponds to Table III above and repeated here to ease comparisons).

%	TCP	DCP	aTCP	aDCP	PP	TCP-QV	Δ%	DCP-QV	Δ%	aTCP-QV	Δ%	aDCP-QV	Δ%	PP-QV	Δ%
10%	0.66	0.80	0.95	0.98	*0.94*	0.79	20%	0.91	14%	0.95	0%	0.98	0%	*0.94*	*0%*
20%	0.52	0.66	0.88	0.96	*0.94*	0.66	27%	0.81	23%	0.90	2%	0.96	0%	*0.94*	*0%*
30%	0.41	0.54	0.81	0.92	*0.94*	0.56	37%	0.71	31%	0.84	4%	0.92	0%	*0.94*	*0%*
40%	0.32	0.43	0.74	0.87	0.94	0.49	53%	0.63	47%	0.78	5%	0.87	0%	0.94	0%
50%	0.25	0.33	0.65	0.82	0.90	0.43	72%	0.54	64%	0.71	9%	0.82	0%	0.91	1%
60%	0.19	0.25	0.57	0.75	0.83	0.38	100%	0.47	88%	0.65	14%	0.75	0%	0.84	1%
70%	0.15	0.17	0.48	0.67	0.66	0.34	127%	0.41	141%	0.57	19%	0.67	0%	0.71	8%
80%	0.09	0.10	0.38	0.57	0.46	0.29	222%	0.34	240%	0.48	26%	0.57	0%	0.56	22%
90%	0.04	0.05	0.26	0.43	0.20	0.26	550%	0.27	440%	0.37	42%	0.43	0%	0.35	75%

Table A1.8. Avg. symmetric difference scores for top-10 results and conjunctive query processing using *1.8M-top1000* training set.

%	TCP	DCP	aTCP	aDCP	PP	TCP-QV	Δ%	DCP-QV	Δ%	aTCP-QV	Δ%	aDCP-QV	Δ%	PP-QV	Δ%
10%	0.66	0.8	0.93	0.98	*0.94*	0.94	42%	0.98	23%	0.97	4%	0.98	0%	*0.94*	*0%*
20%	0.52	0.66	0.86	0.96	*0.94*	0.9	73%	0.95	44%	0.94	9%	0.96	0%	*0.94*	*0%*
30%	0.41	0.54	0.78	0.91	*0.94*	0.86	110%	0.92	70%	0.91	17%	0.93	2%	*0.94*	*0%*
40%	0.32	0.43	0.7	0.85	0.94	0.84	163%	0.88	105%	0.87	24%	0.9	6%	0.94	0%
50%	0.25	0.33	0.6	0.79	0.90	0.81	224%	0.84	155%	0.84	40%	0.86	9%	0.92	2%
60%	0.19	0.25	0.52	0.71	0.83	0.79	316%	0.79	216%	0.79	52%	0.81	14%	0.85	2%
70%	0.15	0.17	0.43	0.61	0.66	0.51	240%	0.58	241%	0.71	65%	0.73	20%	0.72	9%
80%	0.09	0.10	0.34	0.51	0.46	0.28	211%	0.29	190%	0.54	59%	0.60	18%	0.49	7%
90%	0.04	0.05	0.24	0.37	0.20	0.14	250%	0.11	120%	0.35	46%	0.42	14%	0.22	10%

Table A1.9. Avg. symmetric difference scores for top-10 results and conjunctive query processing using *518K-top10* training set.

%	TCP	DCP	aTCP	aDCP	PP	TCP-QV	Δ%	DCP-QV	Δ%	aTCP-QV	Δ%	aDCP-QV	Δ%	PP-QV	Δ%
10%	0.66	0.8	0.94	0.96	*0.87*	0.72	9%	0.86	7%	0.94	0%	0.96	0%	*0.87*	*0%*
20%	0.52	0.66	0.87	0.93	*0.87*	0.58	12%	0.74	12%	0.88	1%	0.93	0%	*0.87*	*0%*
30%	0.41	0.54	0.81	0.9	*0.87*	0.47	15%	0.64	19%	0.82	1%	0.9	0%	*0.87*	*0%*
40%	0.32	0.43	0.73	0.86	0.87	0.39	22%	0.53	23%	0.76	4%	0.86	0%	0.87	0%
50%	0.25	0.33	0.64	0.8	0.85	0.32	28%	0.44	33%	0.67	5%	0.8	0%	0.85	0%
60%	0.19	0.25	0.56	0.74	0.77	0.27	42%	0.36	44%	0.6	7%	0.74	0%	0.78	1%
70%	0.15	0.17	0.47	0.65	0.61	0.22	47%	0.29	71%	0.52	11%	0.65	0%	0.63	3%
80%	0.09	0.10	0.38	0.56	0.43	0.16	80%	0.22	119%	0.43	13%	0.55	-1%	0.47	9%
90%	0.04	0.05	0.26	0.42	0.19	0.12	202%	0.15	200%	0.32	21%	0.42	-1%	0.26	37%

Table A1.10. Avg. symmetric difference scores for top-10 results and conjunctive query processing using *518K-top1000* training set.

%	TCP	DCP	aTCP	aDCP	PP	TCP-QV	Δ%	DCP-QV	Δ%	aTCP-QV	Δ%	aDCP-QV	Δ%	PP-QV	Δ%
10%	0.66	0.8	0.94	0.98	*0.87*	0.87	32%	0.96	20%	0.96	2%	0.98	0%	*0.87*	*0%*
20%	0.52	0.66	0.86	0.96	*0.87*	0.78	50%	0.9	36%	0.92	7%	0.96	0%	*0.87*	*0%*
30%	0.41	0.54	0.78	0.91	*0.87*	0.71	73%	0.84	56%	0.86	10%	0.91	0%	*0.87*	*0%*
40%	0.32	0.43	0.7	0.85	0.87	0.66	106%	0.77	79%	0.81	16%	0.86	1%	0.87	0%
50%	0.25	0.33	0.6	0.79	0.85	0.62	148%	0.71	115%	0.74	23%	0.8	1%	0.85	0%
60%	0.19	0.25	0.51	0.7	0.77	0.58	205%	0.64	156%	0.68	33%	0.72	3%	0.79	3%
70%	0.15	0.17	0.43	0.61	0.61	0.53	253%	0.57	235%	0.6	40%	0.64	5%	0.67	10%
80%	0.09	0.10	0.34	0.51	0.43	0.46	411%	0.47	370%	0.47	38%	0.49	-4%	0.48	12%
90%	0.04	0.05	0.24	0.37	0.19	0.15	275%	0.14	180%	0.32	33%	0.36	-3%	0.23	21%

Table A1.11. Avg. symmetric difference scores for top-10 results and conjunctive query processing using *50K-top1000* training set.

%	TCP	DCP	aTCP	aDCP	TCP-QV	Δ%	DCP-QV	Δ%	aTCP-QV	Δ%	aDCP-QV	Δ%
10%	0.66	0.8	0.93	0.98	0.73	11%	0.89	11%	0.93	0%	0.98	0%
20%	0.52	0.66	0.85	0.94	0.59	13%	0.78	18%	0.86	1%	0.94	0%
30%	0.41	0.54	0.77	0.89	0.48	17%	0.67	24%	0.79	3%	0.9	1%
40%	0.32	0.43	0.69	0.83	0.4	25%	0.57	33%	0.72	4%	0.83	0%
50%	0.25	0.33	0.6	0.76	0.33	32%	0.48	45%	0.64	7%	0.76	0%
60%	0.19	0.25	0.51	0.69	0.28	47%	0.39	56%	0.56	10%	0.68	-1%
70%	0.15	0.17	0.42	0.6	0.23	53%	0.31	82%	0.47	12%	0.59	-2%
80%	0.09	0.10	0.34	0.50	0.16	78%	0.23	130%	0.37	9%	0.48	-4%
90%	0.04	0.05	0.24	0.36	0.12	200%	0.14	180%	0.26	8%	0.33	-8%

Table A1.12. Avg. symmetric difference scores for top-10 results and conjunctive query processing using *10K-top1000* training set.

%	TCP	DCP	aTCP	aDCP	TCP-QV	Δ%	DCP-QV	Δ%	aTCP-QV	Δ%	aDCP-QV	Δ%
10%	0.66	0.8	0.91	0.95	0.68	3%	0.84	5%	0.92	1%	0.95	0%
20%	0.52	0.66	0.84	0.9	0.53	2%	0.71	8%	0.84	0%	0.9	0%
30%	0.41	0.54	0.76	0.86	0.42	2%	0.59	9%	0.76	0%	0.86	0%
40%	0.32	0.43	0.67	0.82	0.34	6%	0.48	12%	0.68	1%	0.82	0%
50%	0.25	0.33	0.58	0.74	0.27	8%	0.39	18%	0.59	2%	0.74	0%
60%	0.19	0.25	0.5	0.66	0.21	11%	0.29	16%	0.52	4%	0.66	0%
70%	0.15	0.17	0.42	0.58	0.16	7%	0.22	29%	0.44	5%	0.58	0%
80%	0.09	0.10	0.33	0.49	0.10	11%	0.14	40%	0.35	6%	0.48	-2%
90%	0.04	0.05	0.23	0.36	0.05	25%	0.07	40%	0.24	4%	0.35	-3%

APPENDIX 2. EXPERIMENTS WITH TEMPORALLY DIFFERENT TEST SETS

To investigate the temporal dimension, we conducted experiments using four new test sets each of which includes 5,000 unique queries. Recall from Section 5 that we have used the first six weeks of AOL query log for training, whereas a set of 1000 test queries were selected from the second six weeks. Now, we split the test queries' time period (i.e., the second six weeks) into three parts: the first set, test-W12, denotes the queries selected from the first two weeks of the test period, whereas test-W34 and test-W56 denote the queries selected from the second and third 2-week periods, respectively. As before, all of these are tail queries (i.e., with frequency 1) that are distinct from the training set. As the fourth query set, we again randomly selected 5,000 queries from a totally distinct log, namely, Excite [Jansen and Spink 2000]. This latter query log spans a totally different time period, i.e., it is collected in December 1999.

In Tables A2.1 and A2.2, we present the results for our best-performing algorithms, namely PP-AAA and PP-AAA-QV family, corresponding to Tables IV and V. For the sake of simplicity, we only report the results for a single pruning level of 90%, whereas the trends are the same for other pruning levels, as well. In the tables, we provide results for the four test sets of 5,000 queries, and the last row simply copies the result of test-1,000 set reported in Tables IV and V, for easy comparison.

Tables A2.1 and A2.2 show that the performance reported on our test-1000 set is almost the same as the other three sets extracted from the AOL log for different time periods, as described above. While the figures over all AOL test sets are comparable, the performance over Excite set is worse than those on AOL. This is not surprising, given that Excite query log covers a totally different time period (i.e., collected in 1999, almost 7 years older than AOL log). To summarize, this experiment indicates that pruned index files created by the combined pruning algorithms preserve the same performance within time, i.e., at least for some reasonable time period.

Table A2.1. Avg. symmetric difference scores for top-10 results and disjunctive query processing. All files are 90% pruned. The last row is copied from Table IV.

Test set	**PP**	**PP-QV**	**PP-TCP**	**PP-TCP-QV**	**PP-DCP**	**PP-DCP-QV**	**PP-aTCP**	**PP-aTCP-QV**	**PP-aDCP**	**PP-aDCP-QV**
Test-Excite	0.25	0.40	0.38	0.44	0.25	0.42	0.26	0.41	0.30	0.41
Test-W12	0.39	0.54	0.51	0.58	0.41	0.56	0.36	0.53	0.43	0.54
Test-W34	0.38	0.52	0.49	0.57	0.39	0.55	0.36	0.52	0.43	0.53
Test-W56	0.38	0.53	0.50	0.57	0.39	0.55	0.35	0.52	0.43	0.53
Test-1000	**0.34**	**0.49**	**0.47**	**0.54**	**0.41**	**0.52**	**0.32**	**0.49**	**0.40**	**0.50**

Table A2.2. Avg. symmetric difference scores for top-10 results and conjunctive query processing. All files are 90% pruned. The last row is copied from Table V.

Test set	**PP**	**PP-QV**	**PP-TCP**	**PP-TCP-QV**	**PP-DCP**	**PP-DCP-QV**	**PP-aTCP**	**PP-aTCP-QV**	**PP-aDCP**	**PP-aDCP-QV**
Test-Excite	0.11	0.23	0.14	0.23	0.11	0.23	0.18	0.26	0.18	0.25
Test-W12	0.21	0.36	0.14	0.33	0.20	0.35	0.33	0.42	0.32	0.41
Test-W34	0.20	0.35	0.14	0.33	0.18	0.35	0.33	0.41	0.31	0.40
Test-W56	0.20	0.37	0.15	0.34	0.18	0.36	0.33	0.42	0.31	0.41
Test-1000	**0.20**	**0.35**	**0.14**	**0.31**	**0.15**	**0.34**	**0.32**	**0.40**	**0.32**	**0.39**

APPENDIX 3. EXPERIMENTS WITH STANDARD EVALUATION METRICS

In addition to average symmetric difference score, we also considered using Kendall's tau (KT) as an evaluation metric to take into account the ranks of the results obtained from the full and pruned index files. However, we experienced several difficulties in practice. First of all, Carmel et al. (please see Section 4.1 in [Carmel et al. 2001]) noted that original Kendall's tau was defined for comparing two permutations, whereas the problem at hand requires comparing two top-*k* lists but not permutations (of the entire documents). Therefore, they adapted a modified KT metric in their work. We also implemented this modification; however soon realized that this metric works reasonably only when both of the compared lists include *k* elements. In contrast, in our setup, it is possible that both the full and pruned indexes yield fewer, i.e., less than *k*, top results. This situation occurs due to several reasons: (i) we experiment with very high pruning levels (i.e., up to 90%), (ii) we also investigate conjunctive processing semantics, which returns smaller number of results on the full and pruned indexes especially for the ODP collection, and (iii) our test queries are distinct from the train (i.e., like cache misses) and indeed each has a test frequency of 1; another reason for these queries having smaller number of results (see Figure 3 in [Skobeltsyn et al. 2008]). As a result, a number of our test queries return less than *k* results on the full and pruned index files, which prevents directly using this modified KT as proposed in [Carmel et al. 2001].

We noticed that the above conclusion is also reached by other researchers (see Section 4.3 in [DeMoura et al. 2005]) as they also take into account the conjunctive query processing. As a remedy, they proposed to add "fake" documents to the result list obtained from a pruned index if there happens to be less than *k* results. This is a partially reasonable solution for this problem, but we observed that if both the full and pruned indexes have yielded less than *k* results for a query, the problem still remains. In this case, we tried to add fake documents to both sides (to the results from the full and pruned index), but we are not convinced that the KT metric works perfectly. As a result, we decided to use symmetric difference score as a metric, which seemed as a less problematic and reliable metric to evaluate our algorithms.

Note that, Garcia proposed to use standard IR metrics for evaluating pruned results (please see Figure 4.2 in [Garcia 2007]). This is possible because the results obtained from the full index can be considered as the ground truth, and the results from the pruned index files can then be evaluated against them using traditional P@10 or MAP. We observed that both metrics (as computed over top-10 results using `trec_eval` software) yield exactly the same trends for comparing the pruning algorithms, but absolute

Table A3.1. MAP (over top-10 results) using disjunctive query processing for test-1000 query set on ClueWeb09-B (relative improvements by QVs are shown in parentheses)

%	PP	PP-QV	PP-TCP	PP-TCP-QV	PP-DCP	PP-DCP-QV	PP-aTCP	PP-aTCP-QV	PP-aDCP	PP-aDCP-QV
60%	0.7668	0.8035 (5%)	0.8161	0.8447 (4%)	0.7525	0.7500 (0%)	0.7097	0.7627 (7%)	0.8123	0.8153 (0%)
70%	0.6407	0.7136 (11%)	0.7933	0.8263 (4%)	0.7294	0.7274 (0%)	0.6931	0.7505 (8%)	0.7856	0.7961 (1%)
80%	0.4697	0.6012 (28%)	0.6679	0.7226 (8%)	0.5985	0.6287 (5%)	0.5918	0.6679 (13%)	0.6423	0.6949 (8%)
90%	0.2391	0.4548 (90%)	0.4189	0.5441 (30%)	0.3529	0.4716 (34%)	0.3722	0.516 (39%)	0.3722	0.5217 (40%)

Table A3.2. MAP (over top-10 results) using conjunctive query processing for test-1000 query set on ClueWeb09-B (relative improvements by QVs are shown in parentheses)

%	PP	PP-QV	PP-TCP	PP-TCP-QV	PP-DCP	PP-DCP-QV	PP-aTCP	PP-aTCP-QV	PP-aDCP	PP-aDCP-QV
60%	0.7291	0.7698 (6%)	0.6544	0.7179 (10%)	0.605	0.603 (0%)	0.7166	0.7602 (6%)	0.8148	0.8149 (0%)
70%	0.5772	0.6661 (15%)	0.6322	0.6993 (11%)	0.5834	0.581 (0%)	0.6976	0.7458 (7%)	0.783	0.7913 (1%)
80%	0.3844	0.528 (37%)	0.5147	0.5984 (16%)	0.4609	0.4852 (5%)	0.5679	0.6369 (12%)	0.6011	0.6604 (10%)
90%	0.1421	0.364 (156%)	0.2606	0.425 (63%)	0.2295	0.3379 (47%)	0.3057	0.4484 (47%)	0.2917	0.447 (53%)

Table A3.3. Avg. symmetric difference scores for *top-1* results and disjunctive query processing (ClueWeb09-B dataset)

%	PP	PP-QV	PP-TCP	PP-TCP-QV	PP-DCP	PP-DCP-QV	PP-aTCP	PP-aTCP-QV	PP-aDCP	PP-aDCP-QV
60%	0.760	0.797 (5%)	0.822	0.852 (4%)	0.768	0.765 (0%)	0.725	0.770 (6%)	0.820	0.820 (0%)
70%	0.630	0.718 (14%)	0.798	0.835 (5%)	0.743	0.747 (1%)	0.707	0.758 (7%)	0.792	0.803 (1%)
80%	0.459	0.602 (31%)	0.664	0.726 (9%)	0.611	0.651 (7%)	0.597	0.671 (12%)	0.635	0.699 (10%)
90%	0.230	0.462 (101%)	0.409	0.551 (35%)	0.348	0.492 (41%)	0.365	0.522 (43%)	0.357	0.521 (46%)

effectiveness scores are higher (because these traditional metrics only consider the “distance” of the pruned results from the “full results”, but not vice versa; in contrast to the symmetric difference metric). In Tables A3.1 and A3.2, we provide the results using MAP as a metric for ClueWeb09-B dataset on our test-1000 query set. P@10 values are almost the same for all cells in the tables (as MAP is also computed over top-10 results) and not reported here to save space. Clearly, the results are in line with those obtained using the symmetric difference score metric (i.e., please compare to Tables VI and VII, respectively).

Finally, we also compute symmetric difference scores for only top-1 result corresponding to cases in Tables VI and VII in Section 6.2. Clearly, the rank of results is not an issue for top-1 result; i.e., the pruned results are either the same as those obtained from the full index, or not. We observe that the gains provided by query views in Tables A3.3 and A3.4 are almost the same as those in Tables VI and VII, respectively.

Table A3.4. Avg. symmetric difference scores for *top-1* results and conjunctive query processing (ClueWeb09-B dataset)

%	PP	PP-QV	PP-TCP	PP-TCP-QV	PP-DCP	PP-DCP-QV	PP-aTCP	PP-aTCP-QV	PP-aDCP	PP-aDCP-QV
60%	0.733	0.782 (7%)	0.726	0.775 (7%)	0.671	0.668 (0%)	0.729	0.767 (5%)	0.820	0.815 (-1%)
70%	0.580	0.684 (18%)	0.704	0.76 (8%)	0.647	0.649 (0%)	0.706	0.752 (7%)	0.786	0.794 (1%)
80%	0.386	0.553 (43%)	0.565	0.648 (15%)	0.512	0.556 (9%)	0.569	0.644 (13%)	0.601	0.672 (12%)
90%	0.155	0.405 (61%)	0.282	0.469 (66%)	0.248	0.40 (61%)	0.310	0.471 (52%)	0.295	0.468 (59%)

APPENDIX 4. EXPERIMENTS WITH TREC 2009 WEB TRACK QUERY SET

For 50 queries (WT09 topic file) used in the TREC 2009 Web Track, we obtained stat mean nDCG (sMNDCG) and traditional MAP/P@10 scores. The former metric was employed in 2009 Web Track and computed using `statAP_MQ_eval_v3` script (available at http://trec.nist.gov/data/web09.html) for top-1000 results. Since the objective in our problem domain is keeping the top-10 results obtained from pruned index files as similar as those obtained from the full index, we also compute traditional P@10 and MAP (for top-10 results only). In particular, we removed the pool inclusion probability column from TREC's official *prels* file (i.e., resulting in binary judgments) and used standard `trec_eval` software to compute traditional P@10 and MAP (see [Kaptein et al. 2010] for a similar approach). The `trec_eval` software is invoked with the parameters "-c -M10" to evaluate top-10 results. These results are presented in Tables A4.1 to A4.6.

Table A4.1. Stat mean NDCG (over top-1000 results) using disjunctive query processing for TREC 2009 Web Track 50-topic query set

%	PP	PP-QV	PP-TCP	PP-TCP-QV	PP-DCP	PP-DCP-QV	PP-aTCP	PP-aTCP-QV	PP-aDCP	PP-aDCP-QV
60%	0.253	0.291	0.291	0.291	0.29	0.29	0.282	0.295	0.292	0.293
70%	0.254	0.289	0.276	0.291	0.275	0.295	0.266	0.3	0.266	0.298
80%	0.222	0.289	0.253	0.291	0.254	0.292	0.244	0.295	0.254	0.292
90%	0.14	0.286	0.222	0.289	0.216	0.288	0.209	0.287	0.212	0.286

Table A4.2. MAP (over top-10 results) using disjunctive query processing for TREC 2009 Web Track 50-topic query set

%	PP	PP-QV	PP-TCP	PP-TCP-QV	PP-DCP	PP-DCP-QV	PP-aTCP	PP-aTCP-QV	PP-aDCP	PP-aDCP-QV
60%	0.0312	0.0346	0.0347	0.0346	0.0343	0.0344	0.0337	0.0343	0.0345	0.0343
70%	0.0293	0.0342	0.0341	0.0346	0.0337	0.0344	0.033	0.0343	0.0333	0.0343
80%	0.0263	0.0344	0.0312	0.0346	0.0308	0.0344	0.0301	0.0342	0.0309	0.0342
90%	0.0154	0.0342	0.0263	0.0344	0.0243	0.0340	0.0255	0.0343	0.0244	0.0342

Table A4.3. P@10 using disjunctive query processing for TREC 2009 Web Track 50-topic query set

%	PP	PP-QV	PP-TCP	PP-TCP-QV	PP-DCP	PP-DCP-QV	PP-aTCP	PP-aTCP-QV	PP-aDCP	PP-aDCP-QV
60%	0.81	0.908	0.91	0.908	0.898	0.904	0.890	0.900	0.904	0.900
70%	0.76	0.898	0.896	0.908	0.884	0.904	0.876	0.900	0.874	0.900
80%	0.684	0.898	0.81	0.908	0.798	0.904	0.788	0.898	0.802	0.898
90%	0.406	0.896	0.684	0.898	0.632	0.892	0.670	0.896	0.634	0.896

Table A4.4. Stat mean NDCG (over top-1000 results) using conjunctive query processing for TREC 2009 Web Track 50-topic query set

%	PP	PP-QV	PP-TCP	PP-TCP-QV	PP-DCP	PP-DCP-QV	PP-aTCP	PP-aTCP-QV	PP-aDCP	PP-aDCP-QV
60%	0.277	0.291	0.291	0.291	0.289	0.293	0.282	0.295	0.292	0.293
70%	0.281	0.288	0.288	0.291	0.286	0.298	0.278	0.300	0.283	0.297
80%	0.263	0.288	0.277	0.291	0.278	0.295	0.267	0.295	0.279	0.292
90%	0.211	0.282	0.263	0.288	0.275	0.286	0.250	0.286	0.270	0.282

Table A4.5. MAP (over top-10 results) using conjunctive query processing for TREC 2009 Web Track 50-topic query set

%	PP	PP-QV	PP-TCP	PP-TCP-QV	PP-DCP	PP-DCP-QV	PP-aTCP	PP-aTCP-QV	PP-aDCP	PP-aDCP-QV
60%	0.0293	0.0346	0.0347	0.0346	0.0341	0.0341	0.0337	0.0343	0.0345	0.0343
70%	0.0274	0.0342	0.033	0.0346	0.0324	0.0341	0.0320	0.0343	0.0321	0.0343
80%	0.0215	0.0343	0.0293	0.0346	0.0287	0.0341	0.0282	0.0342	0.029	0.0342
90%	0.0084	0.0341	0.0215	0.0343	0.0186	0.0339	0.0209	0.0343	0.0188	0.0342

Table A4.6. P@10 using conjunctive query processing for TREC 2009 Web Track 50-topic query set

%	PP	PP-QV	PP-TCP	PP-TCP-QV	PP-DCP	PP-DCP-QV	PP-aTCP	PP-aTCP-QV	PP-aDCP	PP-aDCP-QV
60%	0.760	0.908	0.910	0.908	0.894	0.896	0.890	0.900	0.904	0.900
70%	0.708	0.898	0.872	0.908	0.856	0.896	0.852	0.900	0.846	0.900
80%	0.554	0.898	0.760	0.908	0.744	0.896	0.738	0.898	0.752	0.898
90%	0.236	0.894	0.554	0.898	0.480	0.888	0.546	0.896	0.484	0.896

First of all, please notice that, stat mean nDCG (sMNDCG) score (over top-1000 results as usual in TREC) is similar to that of [Garcia 2009; Table 1] for the full index, revealing the validity/correctness of our experimental setup. Overall, the results are interesting in that the differences among PP-AAA (PP-AAA-QV) algorithms are less emphasized, whereas they all perform better than PP (PP-QV) algorithm. This might be due to small number of test topics, human judgment bias or, especially, query characteristics. In particular, we observed that the TREC query set includes rather easy queries (like “yahoo”, “atari”, “volvo”, etc. with average query length of 2) whereas our test-1000 set includes only tail queries (with frequency 1) from a very large query log (average query length is 3.2 after stopword elimination). Nevertheless, our key finding still holds, i.e., combining PP with other strategies and further with query views considerably improve the performance especially at high pruning levels.