Improved Methods for Static Index Pruning

Wei Jiang*, Juan Rodriguez† and Torsten Suel‡
Computer Science and Engineering
New York University
New York, US
**weijiang2009@gmail.com, †jcr365@nyu.edu, ‡torsten.suel@nyu.edu*

***Abstract*—Static Index Pruning is a performance optimization technique for search engines that attempts to identify and remove index postings that are unlikely to lead to top results for typical user queries. The goal is to obtain a much smaller inverted index that can quickly return results that are (almost) as good as those for the unpruned index. We make two contributions: First, we improve on previous results for pruned index size through a careful analysis of both document and query distribution characteristics. We derive an initial model based on unigram probabilities that obtains gains over previous work in some cases, and a bigram-based approach that achieves some additional improvements. We also devise a simple method for generating query logs in the absence of real-life queries, useful in modeling top results. Our second contribution is to explore, and compare to previously proposed approaches that perform pruning based on how often documents or postings appeared in top positions in the past.**

***Keywords*-static pruning; index; search;**

I. Introduction

Search engines face significant performance challenges due to the huge data sizes and query loads that need to be supported. The largest engines now crawl, analyze, and index trillions of web pages, and process billions of queries per day [1]. One critical cost factor is query processing, which has to scale with both data size and query load, and search engines devote huge hardware and energy resources to this task. There has been lots of research on improving query processing performance, including work on various caching techniques, high-performance index compression, distributed and data-parallel query execution, and a large category of optimization techniques commonly referred to as *Early Termination*, *Pruning*, or *Top-k Query Processing*.

In this paper, we focus on one such optimization technique, *Static Index Pruning*. In a nutshell, the idea is to identify, through suitable analysis of the document collection and query distribution, those index entries (postings) that are most likely to lead to top search results for typical queries, and to drop any other postings from the index. The goal is to obtain a much smaller index that achieves almost the same result quality as the unpruned index, thus requiring much less memory and leading to faster query processing over shorter inverted lists.

To motivate the problem, consider a leading search engine of today.[1] Suppose there are about 4 trillion ($4 \cdot 10^{12}$) documents that have been indexed, with each document having about 250 index entries on average, or a total of 10^{15} postings. Suppose the engine receives 5 billion queries per day with 3.4 terms per query. This implies that at most 170 billion postings in the index lead to a top-10 result per day, or at most $5.1 \cdot 10^{12}$ per month – in fact far less once we consider repetition of queries and postings. Thus, more than 99.5% of all postings do not result in even a single top-10 result in a given month. While we obviously cannot reliably identify the 0.5% of the index that will be used in the next month, we might hope that we can identify a somewhat larger subset of the index, say 10%, that contains most of these postings and achieves comparable result quality as a full index on common measures of effectiveness.

There has been a fair amount of previous work on Static Index Pruning [2]–[9], mostly based on ideas such as keeping all postings above a global impact threshold, keeping the highest scoring postings in each list or each document, or keeping postings or entire documents that were part of top results in past queries. While this has led to some promising results, we believe that further improvements are possible. Thus, our goal here is to build on this work, and to derive a model that can combine these and other ideas to achieve better trade-offs between index size and result quality according to standard measures of retrieval quality. We note that we are dealing with a very feature-rich environment, with many features that could be used to estimate how useful a posting is, i.e., how likely it is to lead to top results for likely queries. Thus, we are considering pruning as a prediction problem, where we can use suitable statistical techniques (language modeling, machine learning) to decide which postings to keep.

The remainder of this paper is organized as follows. In the next section, we provide some background on indexing and pruning, and discuss related work. Section III summarizes our contributions, and Section IV provides technical details about our approach. Our experimental results are presented in Section V. Section VI provides some concluding remarks.

[1]The following numbers are rough estimates based on public sources and some discussions.

II. Background and Related Work

In this section, we first provide some background on inverted index structures, ranked queries, index pruning, and early termination techniques. We then discuss previous work related to static index pruning. For additional details on indexes and queries, we refer to [10,11].

A. Background

Inverted Indexes: Suppose we have a collection D of n documents $d_0, d_1, \ldots d_{n-1}$ where each d_i is a sequence of words (terms). Then an *inverted index* for D contains one *inverted list* L_t for each distinct term t that occurs anywhere in D. Each inverted list L_t is a sequence of postings, where each posting p contains information about the occurrences of term t in some document d_i.

We assume that a posting contains the document ID (docID) i of d_i, and either the number of occurrences of t in d_i (called frequency) or a precomputed impact score of term t in d_i. Thus, a posting is of the form (i, f) or (i, s). Postings in each list are usually sorted by their docIDs and stored in suitable compressed form, but this is not central to our work here. Also, postings could contain additional data such as information about the contexts and positions of the term occurrences in the documents.

Queries and Ranking: We assume that a query is a set of terms, and that the goal of the system is to return the best top-k results to the users, say for $k = 10$ or 100. This is done using a ranking function $r(d, q)$ that returns a score for a document d with respect to query q. In particular, we assume that for a query q, we first perform a Boolean filter, typically a conjunction (AND) or disjunction (OR) of the query terms, and then compute the score $r(d, q)$ for those documents that pass the Boolean filter. Throughout this paper, unless stated explicitly otherwise, we assume disjunctive top-k queries with $k = 10$.

In this paper, as in previous work on static index pruning, we assume a simple ranking function that can be computed directly from the inverted index postings of the query terms. In particular, our experiments use the widely used BM25 ranking function. We note that while current search engines use highly complex ranking functions with hundreds of features, they usually rely on a very simple function to perform a first pass over the inverted lists to identify a set of promising documents that are further filtered and reranked using the full ranking function. Performing index pruning directly for a given complex ranking function is an interesting problem for future research, but has not been considered in previous work.

Index Pruning: We now discuss the static index pruning problem in more detail. Given an inverted index, the goal is to prune as many postings as possible from the index while still returning good top-k results on incoming queries. More precisely, we have an inverted index and a set of training queries, plus potentially other data sources (e.g., n-gram collections). We can analyse this data to decide which postings to keep and which to prune, and then the resulting pruned index is evaluated on a set of testing queries kept separate from the training queries. A good pruning scheme is one that obtains a good trade-off between the size of the pruned index and the quality of the results.

As in previous work, we mostly focus on the percentage of postings kept as our measure of index size. We note that this is not the same as physical index size due to the common use of index compression techniques. Pruning widens the docID gaps between successive postings and may thus adversely impact compression; also, pruning policies may decide to keep more of the shorter or more of the longer lists. Index size is also not a reliable predictor of increased query processing speed, as we may decide to keep more postings in the frequently accessed inverted lists. On the other hand, if most of the unpruned index is kept on disk, while the pruned index fits in main memory, the observed speed-up could be much larger than the reduction in size.

We also need measures to judge the quality of the query results obtained by the pruned index. There are two common ways to do this: (1) we can look at what percentage of top-k results returned by the unpruned index is still returned after pruning (or a closely related measure such as the symmetric difference between pruned and unpruned results). (2) We can use traditional IR measures such as P@10, MAP, or NDCG that are more lenient in that they do not require the exact same results to be returned. In our work, we evaluate our techniques on both types of measures. However, we use the first type, percentage of results kept, and a related measure, percentage of result postings kept in the pruned index, as our objective during the design of our pruning schemes, as it is easier to reason algorithmically about these.

Early Termination Techniques: Static index pruning is closely related to a class of optimization techniques for IR query processing often referred to as *early termination* (ET). Formally, an exhaustive query processing algorithm is one that completely evaluates the given ranking function on all documents that pass the initial Boolean filter, and any non-exhaustive algorithm uses early termination. A large number of ET algorithms have been proposed in the literature. This includes algorithms that skip over large parts of the inverted lists during query processing [12]–[19], index tiering methods that build separate indexes for high- and low-quality documents and then evaluate most queries only against the high-quality one [9,20,21], and cascading techniques that first use a simpler approximation of the full ranking function to identify candidates for further evaluation [22,23]. Static index pruning differs in that the decision on what postings to skip is done ahead of time during index construction.

We call a method *safe* [17] if it is guaranteed to return exactly the same results as the exhaustive method. Many of the above algorithms are safe, but tiering, cascading, and

static index pruning methods are unsafe. For static index pruning this seems necessary, since for any posting we should be able to construct a (maybe very unlikely) query containing the posting term that has this document in the top results. Thus, if we require safety, no pruning is possible. Unsafe techniques are usually evaluated in terms of either standard IR quality measures or the percentage of original exhaustive results still returned.

Early termination is widely used in current search engines, and is necessary to support billions of queries per day. Several ET techniques are commonly used in combination; in particular, some forms of cascading and index tiering are probably used by all major commercial search engines. On the other hand, static index pruning appears to be less widely used, but we think that improvements in the techniques could change this. Note that static index pruning, as many of the ET techniques, should improve with collection size for constant k (number of results returned); unfortunately our experiments are limited to a few ten million documents.

B. Related Work

Static index pruning was first introduced by Carmel et al. [3], and was subsequently studied by a number of other researchers [2,4]–[9,24]–[30]. We now discuss this work.

Index pruning techniques can be divided into three sets. Posting oriented pruning techniques that decide whether to keep individual postings, document oriented pruning techniques that keep or delete complete documents and list oriented pruning techniques that keep or remove the inverted lists.[2] We can further cluster previous methods into three classes, as follows:

Rank- and Impact-Based Methods: Several papers consider posting pruning rules based on the impact or within-list rank of a posting according to some given ranking function, e.g., BM25 or a Cosine measure. That is, rules are designed to try to preserve many of the top results obtained by that ranking function on the full index. In particular, Carmel et al. proposed two methods, UP, which uses a global cut-off on the impact scores of postings, and TCP, which selects the highest impact postings from each inverted list. Büttcher et al. [25] also evaluated TCP as part of TREC 2006, showing promising results. Work by Chen and Lee [7] revisited the UP method, and provided a theoretical foundation for it. More recent work by Chen et al. [8] improves on [7] by refining the mathematical foundation and optimization objective. Nguyen [26] showed how to improve pruning by combining a number of features to determine which postings should be kept.

Optimizing Pruned Retrieval Quality: Another set of approaches tries to select postings for the pruned index such that retrieval quality, measured in terms of measures such as P@10 or MAP, is optimized. Thus, in contrast to the first set of techniques, there is no given ranking function whose results should be preserved; instead the idea is to try to directly identify and keep "good stuff". An example is the work by Büttcher and Clarke [4], which selects postings based on KL-divergence; the idea is to select those postings in a document that are most likely to result in the document being highly relevant under a query. Approaches by Blanco and Barreiro [5], de Moura at al., and Thota and Carterette [29] use language models to select good postings to keep. Yet other approaches try to remove entire documents [28,30] or inverted lists [24,31] that are unlikely to be useful.

Using Query Traces: Another natural idea is to use past queries to decide which postings should be kept. Work by Lam et al. [2] combines the impact-based approach in [3] with a query-based approach that looks at how often a posting appears in the result set. Altingovde et al. [6] introduce an approach called QueryView (QV) that marks and keeps postings that were part of top results in past queries. Another version combines this rule with the TCP method in [3]. Work by Anagnostopoulos et al. [9] applies a similar idea to complete documents, by keeping documents that were often returned as top results in the past.

One advantage of pure query-trace based approaches is that they make no assumptions about the ranking function, i.e., that it consists of term-wise components. Instead, they treat ranking as a black box and only use the results. One disadvantage is that they require the execution of large numbers of queries. This can be a problem because few large query traces are available to academic researchers, and also because methods that select postings from previous top results fundamentally do not scale well. This is because the cost of pruning to a fixed fraction of the index grows with the square of the document collection, as both the number of queries needed to mark enough postings as top results, and the cost per query, usually grow linearly with the collection size.[3] This problem is slightly less pronounced for methods such as [9] that mark complete documents.

Comparison to our Work: Our main approach relates mostly to the first and last categories. Our goal, or at least our intermediate goal during algorithm design if not during final evaluation, is to preserve top results for a given ranking function such as BM25. We are building on Nguyen's work [26] in that we also try to identify additional posting features that can predict good postings, though we believe statistical techniques are best for the actual prediction. We also build on query-based approaches, and in fact use of queries is central to our methods and their performance.

Our basic approach attempts to estimate the likelihood that a posting will lead to a top result given a random query; we call this the *promise* of a posting. This is actually closely

[2]Note that spam removal can be seen as falling in the latter category, but most pruning techniques try to go significantly beyond spam removal.

[3]Some early termination techniques such as [32] use asymptotically less than linear time, but are rarely used in search engines.

related to the query-based approach in [6] as follows: if we could execute an infinite number of training queries and count how often each posting occurs in the top-k, then after normalizing these counts would converge to the likelihood we are trying to estimate.

We also compare our approach to the approaches in [6] and [9]. Since we do not have access to enough queries, we implement a language model similar to [33] to generate artificial queries that are then used for pruning. As discussed above, convergence of such methods is very slow as each query only marks about $k \cdot Q$ postings (or k documents) as top-k, where Q is the average query length, and indexes commonly have many billions or trillions of postings and documents. Thus, the costs of such methods are very high, and might be more than what is saved by pruning. This is also why, as shown in [2,6], hybrid approaches that incorporate impact- or rank-based heuristics outperform pure query-based ones when the number of queries is limited.

Our method basically avoids this problem by generalizing – rather than just picking past top postings, we also pick other postings that look similar to them according to various features. As a result, we get very good performance on large collections while using only a few ten thousand training queries. When comparing our performance with the ones based purely on queries [6,9], it also makes sense to think about it as a complementary method – if there are not enough queries or enough time to (preferably repeatedly) mark a sufficient number of postings or documents, one could fill up the index with other similar postings using our approach.

III. Our Contributions

In this paper, we study static index pruning methods that attempt to achieve a good trade-off between index size and retrieval quality. Our main contributions are as follows:

- We describe an approach that models the likelihood of a posting leading to top results under a given query distribution. The approach uses collection properties and a set of training queries to obtain good estimates.
- We describe several algorithms that use the approach, including a basic unigram-based estimator, a method based on a bigram model of the query distribution, and extensions of these methods that try to cover top results with postings. All methods can be efficiently parallelized and are suitable for a MapReduce environment.
- We perform an experimental evaluation on two large collection, GOV2 and ClueWeb09 CatB, that shows some improvements in the size-quality trade-off over previous work.
- We compare our methods with the approaches proposed by Altingovde et al. [6] and Anagnostopoulos et al. [9], which select postings and documents based on past top results. We show that an approach that uses large numbers of artificial queries based on a language model can do extremely well, though at a significant preprocessing cost.

IV. Our Pruning Algorithms

In this section, we describe our approaches for static pruning. Recall that we are given an inverted index for a document collection D and a training set of queries Q. Our objective in the algorithm design is to retain postings such that we maximize the expected number of top-k results on the full index that are retained under a randomly chosen testing query.

In the following, given a posting p we use $t(p)$ to denote the term indexed by p, $len(p)$ to denote the length of the inverted list containing p, $doc(p)$ to denote the document that p belongs to, and $rank(p)$ to denote the rank of p in its inverted list (where rank zero means that p has the highest impact score in its list). Also, given a query q, we say that a posting p leads to a top-k result d_i for q if $t(p) \in q$ and $doc(p) = d_i$.

This section is organized in four parts. First, we present a basic unigram approach, and a more advanced bigram approach, that attempt to maximize the expected number of postings in the pruned index that lead to a top-k result under a random query on the full index. However, maximizing the number of top postings retained does not necessarily maximize the number of top results retained by the pruned index, as discussed below. To address this issue, we then describe a refinement of the approach that selects postings to maximize the number of top results *covered* by postings. We then discuss how to efficiently implement this approach. Finally, we describe a query-log approach based on [6] and [9].

To understand the difference between the two goals above, maximizing top postings and top results, consider a 3-term query $q = a, b, c$ and three postings (one from each list) leading to the same top-k result d_i. If one of these postings is not included in the pruned index, then the pruned index cannot return d_i in the conjunctive query case. Even for disjunctive queries, the chances of d_i still being in the top-k in the pruned index are greatly reduced. Hence, our goal should really be to maximize the number of top-k results completely covered by postings in the pruned index. Thus, while selecting postings, it matters what other postings are in the pruned index. A posting becomes more attractive for selection if a lot of its potential partner postings (postings leading to the same top-k result under likely queries) are in the pruned index. Our refinements, presented in Subsection IV-B, take a simple greedy approach based on this insight.

A. Maximizing Retained Postings

The basic idea in this approach is very simple. Let $p \rightarrow k$ denote the event that a posting p leads to a top-k result. Our approach is to estimate for each posting p the probability of $p \rightarrow k$ under a random query, and to then select postings for the pruned index based on this estimate. Observe that

$$Pr[p \rightarrow k] = Pr[t(p) \in q] \cdot Pr[p \rightarrow k | t(p) \in q]. \quad (1)$$

We will now show how to separately estimate these two terms. For the second term, we show two estimations, one based on a unigram model, and one based on bigrams. Given such estimates and a posting p, we refer to the estimated value of $Pr[p \rightarrow k]$ as the *promise* of p.

Estimating $Pr[t(p) \in q]$: The solution for this part is straightforward. Using the training query set, we derive the probability using suitable language modeling techniques. In fact, a simple approach based on Good-Turing Smoothing (see, e.g., [34]) performs quite well for unigrams. As we only have fairly small query traces available, slight improvements can be obtained using Interpolated Kneser-Ney Smoothing [35] where we also interpolate with a document model.

Estimating $Pr[p \rightarrow k|t(p) \in q]$ (unigram case): Our first approach, which we call UPP (unigram posting promise), uses simple statistical techniques to estimate the probability of a posting p being in the top-k, given that the query contains $t(p)$. This is done by following an approach proposed in [33], where a small set of training queries is used to predict this probability. In particular, we create a training set as follows: for each query, and each posting in the disjunction of the query terms, we store a set of posting features and a label – 1 if the posting led to a top-k result on the query, and 0 otherwise. Then we use this data to learn to predict how likely a posting is to lead to a top result one a query containing the posting term.

We considered a number of different posting features, including the impact score, $len(p)$, $rank(p)$, the relative rank $rank(p)/len(p)$, the size of the document, and query-related features such as the average length of queries containing $t(p)$ or the likelihood that p belongs to the shortest or longest list in q. We also tried several prediction methods including logistic regression and regression trees. In the end, we found that simply using the relative rank and $len(p)$ did very well, and that a simple bucketing of training data along the two axes, followed by nearest neighbor smoothing for sparse buckets, was sufficient and fast given the large amount of training data obtained from the queries. That is, we created a few dozen classes of list lengths (e.g., lists shorter than 100, lists between 100 and 120, lists between 120 and 144, etc.) and a few dozen classes of relative ranks (e.g., above 0.5, between 0.25 and 0.5, etc.), and then stored for each cell the ratio of positive to all examples. Note that during pruning, we have to get a prediction for every posting in the index, and thus lookup speed is very important; here, we just need a lookup into a small 2-D table.

Estimating $Pr[p \rightarrow k|t(p) \in q]$ (bigram case): The above case takes a unigram approach in that it considers a posting p without looking at other terms in the same document that might give a boost to p. Suppose many queries that contain "cat" also contain "mouse" or "food". In that case, a posting for "cat" located in a document that also contains "mouse" and "food" would have a higher chance of leading to a top result than a document not containing these terms – particularly when the "mouse" and "food" postings have high impact scores. Thus, a better approach should model term co-occurrences in queries and documents. We call this approach BPP (bigram posting promise), and it is based on the following estimate:

$$Pr[p \rightarrow k|t(p) \in q] \approx \sum_{p' \in doc(p), p' \neq p} (Pr[t(p') \in q|t(p) \in q] \cdot Pr[p \rightarrow k|t(p) \in q \cap t(p') \in q]) \quad (2)$$

Thus, we estimate $Pr[p \rightarrow k|t(p) \in q]$ by summing over all other postings in the document. For each other posting p', we take the probability that its term appears in a query already containing $t(p)$, multiplied by the probability that p (and thus also p') leads to a top result on such a query. (Note that this approximation assumes that both query terms need to exist for a document to score in the top.)

To implement this, we need to estimate $Pr[t \in q \cap t' \in q]$ by building a bigram model on the training queries using interpolated Kneser-Ney. We also need to estimate $Pr[p \rightarrow k|t(p) \in q \cap t(p') \in q]$. This is also done in a similar way as for the unigram model. Now we have two postings p and p' that each have features. We again use the list length and relative rank features, and use the training queries to learn a 4-D lookup table for $Pr[p \rightarrow k|t(p) \in q \cap t(p') \in q]$. More details on the implementation are given in Subsection IV-C.

B. Maximizing Retained Results

As explained before, one problem with both UPP and BPP is that they are focused on maximizing the expected number of retained top postings rather than top results. One possible fix for this is to change the definition of the promise of a posting by considering which other postings from the same document have already been added to the pruned index. Thus, choosing one posting from a document increases the promise of other postings in the document that have not been chosen yet, as it increases the likelihood that multiple (or all) postings of a top result are covered. For the unigram model, this is achieved by redefining the promise of p as

$$Pr[t(p) \in q] \cdot Pr[p \rightarrow k|t(p) \in q] \cdot (1 + \alpha \cdot SPI(doc(p)))$$
$$SPI(d) = \sum_{p' \in PI(d)} Pr[t(p') \in q]. \quad (3)$$

$SPI(d)$ is the set of postings from a document d that have already been selected into the pruned index, and α is a weight determining how much we want to boost the promise. Note that $SPI(d)$ is initially zero, and is updated whenever we pick a posting from d. So we basically increase the promise of postings that not only have a good chance of leading to a top result, but where many possible partner postings (under a unigram model of co-occurrence) have already been picked, thus encouraging the algorithm to pick most or all postings for top results. Note that postings are not guaranteed to be picked in monotonically decreasing

order of promise anymore, since the promise of unpicked postings increases whenever we pick a posting from the same document. But the basic intuition is sound: given two postings with the same unboosted (UPP) promise score, we should pick the one from the document where many common query terms have already been picked.

This idea can be easily extended to the bigram approach, by adding to the promise according to BPP (2) an additional term:

$$Pr[t(p) \in q] \cdot \alpha \cdot \sum_{p' \in PI(doc(p)), p' \neq p} (Pr[t(p') \in q | t(p) \in q] \cdot Pr[p \rightarrow k | t(p) \in q \cap t(p') \in q]) \quad (4)$$

Note that this basically gives extra weight, as determined by α, to promise derived from those partner postings p' that have already been picked into the pruned index. As in the unigram case, a postings promise increases over time as other postings are picked from the same document.

We refer to these two methods for boosting the promise as UPP-α and BPP-α, respectively. Note that UPP-0 and BPP-0 are the basic methods without boost.

C. Efficient Implementation

We now describe how these methods can be efficiently implemented. The crucial observation is that the order in which the postings in one document are selected into the pruned index does not depend on any other document, and does not depend on how many postings are picked overall. Our implementation consists of the following steps:

- **Language Modeling:** We build a query language model based on the training queries and a sample of the documents, using interpolated Kneser-Ney based on the library in [36]. For bigrams, we observed that a linear interpolation between a query and a document model does not work well, as it tends to overestimate the frequency of pairs that are very common in the collection but rare in the query trace. Thus, the pruned index would contain too many such postings. Instead, we binned bigrams into 40 classes according to their frequencies in the training queries and documents, and did a separate linear interpolation in each bin to minimize perplexity. This step is very efficient and can easily be done on a single machine.
- **Learning the Tables:** Here, we execute the training queries on our collection, and use the results to derive the 2-D table for the unigram and the 4-D table for the bigram model. For the unigram model, for each query, and each posting in a list participating in the query, we check if the posting makes it into the top-10 results. This gives us for each cell in the table the probability whether a posting falling into the cell will make it into the top-10. For the bigram model, for each query, and each pair of postings in lists participating in the query, we do the same. However, to limit the impact of long queries with many term pairs, we weighted by the number of term pairs. Afterwards, we smoothed the resulting 2-D and 4-D tables to deal with sparse cells. This step is also highly efficient and mainly involves executing a few ten thousand queries and then updating the counts in the cells for each posting or pair, which takes a few hours.
- **Document Analysis:** This step is the most expensive, but is easily performed in a MapReduce environment, as each document can be analyzed in isolation. For each document, we determine the order in which postings would be picked into a pruned index, and their promise score at the time they are picked. In UPP and BPP, this involves computing the promise of each posting and then sorting in decreasing order of promise. For UPP, the cost is clearly $O(n \lg n)$ where n is the number of postings in the document. However, for BPP the cost becomes $O(n^2)$ as for each posting we need to sum over all other postings in the document. Note that BPP also requires n^2 lookups into the 4-D table and the language model.
  For UPP-α and BPP-α, we first compute the initial promise of each posting as in UPP-0 and BPP-0. Then we repeatedly pick the posting with highest promise, and update the promises of all unpicked postings in the document. For UPP-α, this update can be done by simply updating the $SPI(d)$ value of the document, since the order in which postings are selected within a document is not changed compared to UPP-0; thus complexity stays $O(n \lg n)$. For BPP-α, for each posting p' picked, we need to loop over all unpicked postings p and increase its promise by $Pr[t(p) \in q] \cdot \alpha \cdot Pr[t(p') \in q | t(p) \in q] \cdot Pr[p \rightarrow k | t(p) \in q \cap t(p') \in q]$. The total cost is again $O(n^2)$.
- **Posting Selection:** The input of this step is, for each document, a sequence of postings in the order they should be picked, and the promise score of each posting at the time it is picked. For UPP-0 and BPP-0, this step is just a selection operation, where we pick the postings with the highest promise across all documents. For UPP-α and $BPP-\alpha$, where promise is not monotonically decreasing, we use a greedy approach using a heap, with promise value as key. We pick the posting with the highest promise overall and replace it by the next posting from the same document, until enough postings have been picked. (We note that we can use other approaches that try to look ahead into the promise curve of a document, but this does not seem to change results measurably.) This step is also very fast.

D. Simple Query-Log Based Approaches

As described before, work in [6] and [9] describe query-log based approaches for posting pruning and document pruning, respectively. The basic underlying idea in both approaches is to count for each posting, for each document, the number of *hits* (i.e., how often it has appeared in top

Table I

Dataset	GOV2	Clueweb09 B
# postings	6,451,948,010	17,075,485,964
# terms	37,728,619	86,532,822
# documents	25,205,179	50,220,423
avg document length	256	340

results in the past), and use this to retain the right items. Given a large enough sample of queries, these methods can obtain a good estimate of the likelihood that a posting or document is in the top-k for a random query (by dividing the hits by the number of queries). However, this requires a lot of queries, and computation to execute the queries. This is most severe for the posting pruning method, and [6] addresses this by keeping any posting that has been a top result even once, by looking at top-1000 results to achieve better coverage, and by blending the approach with other methods.

For our data, we have a much more limited query set, making a comparison difficult. To allow a comparison to [6] and [9], we use the toolkit in [36] to create a query language model as described in [33], to generate and execute large numbers of artificial queries, and finally look at the number of hits to decide what to keep (and in some case normalizing these scores suitably). We refer to these methods as doc hits and post hits.

V. Experimental Results

In this section, we provide experimental results evaluating our proposed UPP and BPP methods, the doc-hits and post hits query-log based methods in [6] and [9], and some previous techniques.

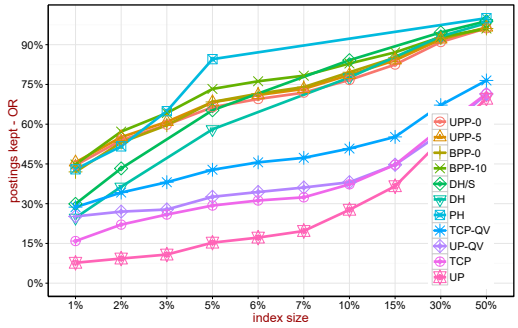
We used GOV2 and Clueweb09 CatB as our experimental dataset, but due to the space limitation of this paper, we show most of our results based on GOV2, while providing some selected results for Clueweb09 Cat B. For the experiments with GOV2 dataset, we report $P@10$ results using TREC topic queries 701 to 850, with average query length 3.11. We also utilized the 100K queries from the TREC 06 efficiency task track, with average query length 4.11 terms. We use 95% of the queries for training, and 5% for testing. We use these testing queries to report results on how many top postings and results from the unpruned index are kept, while training queries are used for computing language models and lookup tables. For the experiments with ClueWeb09 CatB, we used 200 judged topic queries from the TREC Web Tracks 2009 to 2012, with an average query length of 2.5 terms, to report $P@10$. An additional 174K queries from the TREC Million Query Track, with an average query length being 3.56 terms, were used for training (95%) and testing (5%). We use partial BM25 scores as posting impact score. Table I summarizes the statistics of these datasets. A discussion about the methods used in this paper and the way we generalize query view and query covering follows:

- **Uniform Pruning (UP)**: A global impact score threshold is computed for each level of index size, and only postings above the threshold are included in the pruned index.
- **Term Centric Pruning (TCP)**: This method computes an impact score threshold per inverted list, and keeps only postings above the thresholds. The thresholds are chosen by sorting each inverted list by their impact score, and choosing the appropriate level accordingly.
- **QueryView (QV)**: We use the training query set and collect the set of top-100 postings returned. This set of postings form the query view. These postings are "protected", and thus will not be pruned.
- **Uniform Pruning with Query View (UP-QV)**: We first perform QV, and then apply a uniform pruning (UP) method to choose the rest of the postings in the pruned index.
- **Term Centric Pruning with Query View (TCP-QV)**: We first perform QV, and then apply a term centric pruning (TCP) method to choose the rest of the postings in the pruned index.
- **Doc Hits (DH)**: We use a large amount of artificial queries generated by our language model to collect the top-10 results and the hit frequency of documents. We then select documents based on decreasing order of hit frequency.
- **Doc Hits/Doc Size(DH/S)**: This method modifies DH by dividing the number of hits by the size of the document, to give priority to smaller documents.
- **Posting Hits (PH)**: We use a large amount of artificial queries generated by our language model in order to collect the top-10 posting hits. We then select postings based on decreasing order of the hit frequency.

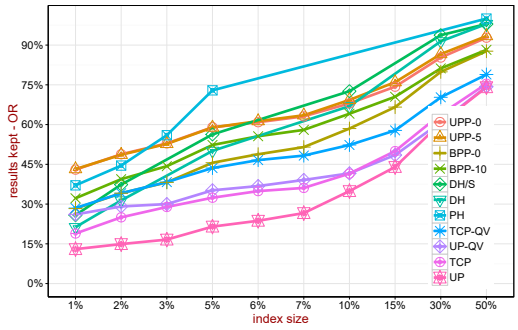
Unless explicitly stated otherwise, all experiments are for top-10 results under OR semantics. We mostly focus our work on cases where we prune 90% or more of all postings, which we think is the most interesting range, though we also provide numbers for a few larger pruned indexes.

Figure 1 shows that UPP-α outperforms the four baseline (UP, TCP, UP-QV, TCP-QV) methods by a large margin, for all three evaluation metrics (top postings kept, top results kept, and $P@10$). In particular, we see that we can prune more than 90% of the index and still get results for $P@10$ that are close to those for the unpruned index. As expected, DH, DH/S, and especially PH, do quite well in all three evaluation metrics. PH does best overall for results and postings kept, and at low index sizes for $P@10$, but as discussed above, these methods have limitations —the significant amount of queries needed in order to get a precise estimation of doc or posting hits, and the high computational power required to achieve good coverage. For cases where either the query size or the computational effort required to measure hits is limited, UPP and BPP methods are preferred.

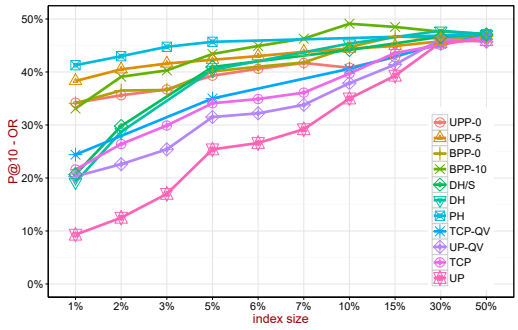
Next, in Figure 2 we look at the UPP-α methods and show the effect of varying α under OR and AND semantics, in order to increase the number of results kept. We show no real



(a)

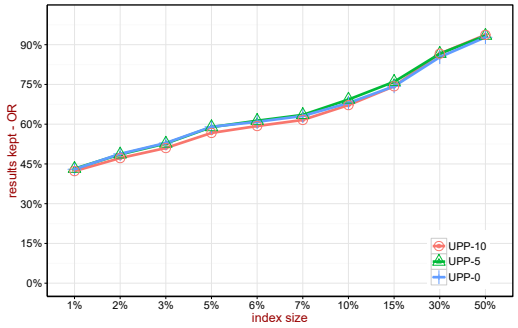


(b)

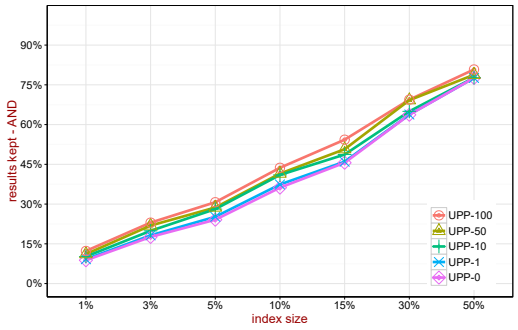


(c)

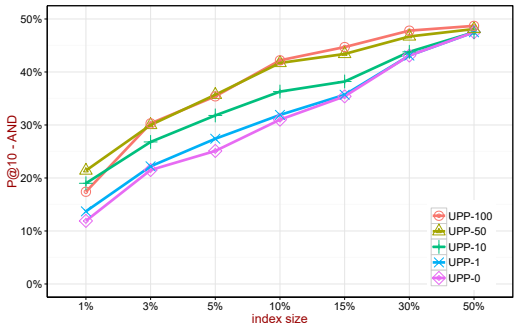
Figure 1. Quality measures for all methods



(a)



(b)

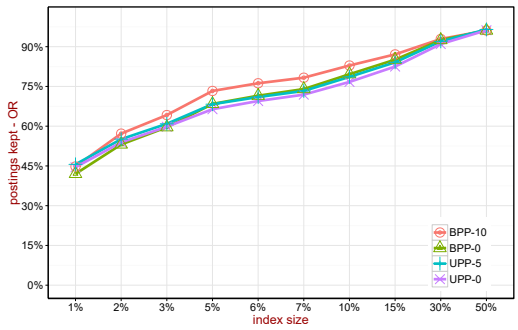


(c)

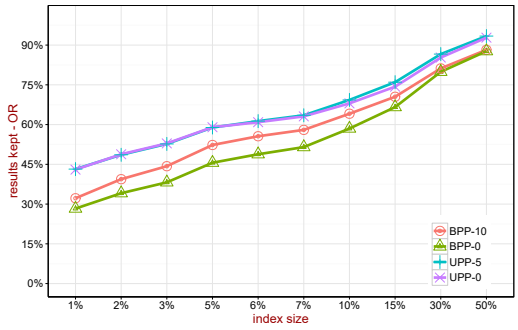
Figure 2. Quality measures for UPP at varying values of α

improvement for results kept under OR semantics, but there is a visible improvement for AND queries, which require all postings leading to a top result to be kept in the pruned index. For $P@10$, Figure 2 also shows a significant boost of the UPP-α methods over UPP-0. We also observed a smaller but still clear improvement in $P@10$ for OR queries (shown as part of the next figure). Overall, the α refinement appears to perform what it was designed for, though we were hoping for clearer improvements for the OR case.

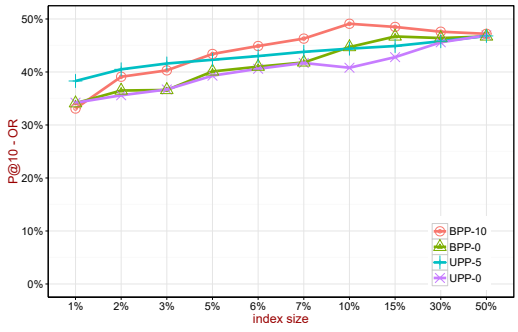
In Figure 3 we compare UPP and BPP via varying α on all three evaluation metrics. BPP does slightly better than UPP on the number of top postings kept, confirming our intuition. However, for top results kept UPP outperforms BPP, while for $P@10$ there are improvements under BPP, especially as the index size grows. The behaviour for top results kept can be explained by a drawback of the BPP method: it tends to promote postings in longer lists but with lower impact factors into the pruned index, based on their popularity as partners for other higher scoring posting. (The average impact score of postings selected by BPP was significantly lower than for those selected by UPP.) These postings are often not essential, in that the higher scoring partners would be able to make it into the top results without these postings. To fix this problem, a correction factor which we call essentiality factor of a posting has been introduced. The essentiality factor helps lower the promise of low-impact postings, thus hopefully mitigating this problem, but we were unable to obtain results in time for this submission. We also suspect that results for BPP are still hampered by limits of the bigram language model. One basic problem here is the limited amount of training queries we have available, as many bigrams never occur



(a)



(b)



(c)

Figure 3. Quality measures for UPP and BPP methods

in the training queries. A slight improvements may still be possible on the currently available queries, using better, possibly non-linear interpolation with the document model. We also note that the document analysis phase of BPP, as described in Section IV-C, with a cost per document that is quadratic in its number of postings, took a significant amount of computational resources (a few hundred CPU core-hours for the GOV2 collection with 25M documents), though, as noted before, this work is highly parallelizable. We also believe we can improve BPP by, e.g., only considering important posting pairs (say, with bigram probability above a threshold), but this is left for future work. In contrast, the cost of the UPP methods is only moderately larger than that of indexing.

Finally, we show some selected results for the ClueWeb09 CatB data set. Table II shows that on this data set, our methods again, attain some improvements over previous methods. Note that the $P@10$ results are generally lower for ClueWeb09 than for GOV2, as the queries are considered tougher.

In summary, our experimental results show that our methods can outperform previous methods and achieve fair results even if more than 90% of all postings are pruned away. Previous works have also been presented and it is shown to perform well, with the limitation that significant amounts of queries and computational expense are required.

VI. Conclusions

In this paper, we have proposed several new algorithms for static pruning of inverted indexes and a comparison with query view [6] and query covering [9]. Our approach attempts to estimate the likelihood that postings result in top results, based on various posting features and collection and query trace statistics. Our experimental results show that the new algorithms achieve measurable improvements over previous work.

There are several extensions that we plan to include in the extended version of this paper. This includes experiments with other bigram language models that might obtain slightly better pruning, and other extensions and optimizations of our approach such as hybrid unigram/bigram methods and the use of more posting features for estimating $Pr[p \rightarrow k]$. We plan to also study the index size / query cost trade-off under different cost models and for actual query processing algorithms.

Acknowledgement This research was supported by NSF Grant IIS-1117829 Efficient Query Processing in Large Search Engines, and by a grant from Google.

References

[1] Google, "How search works: The story inside search," http://www.google.com/insidesearch/howsearchworks/thestory/.

[2] H. T. Lam, R. Perego, and F. Silvestri, "On using query logs for static index pruning," in *Proceedings of the 2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology - Volume 01*.

[3] D. Carmel, D. Cohen, R. Fagin, E. Farchi, M. Herscovici, Y. S. Maarek, and A. Soffer, "Static index pruning for irsystems," in *Proc. of the 24th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval*, 2001.

[4] S. Büttcher and C. Clarke, "A document-centric approach to static index pruning in text retrieval systems," in *Proc. of the 15th ACM CIKM*, 2006.

[5] R. Blanco and A. Barreiro, "Probabilistic static pruning of inverted files," *ACM Transactions on Information Systems*, vol. 28, 2010.

[6] I. Altingovde, R. Ozcan, and O. Ulusoy, "Static index pruning in web search engines: Combining term and document popularities with query views," *ACM Transactions on Information Systems*, vol. 30, 2012.

[7] R. Chen and C. Lee, "An information-theoretic account of static index pruning," in *Proc. of the 36th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval*, 2013.

Table II
QUALITY MEASURES FOR UPP AND BASELINES ON THE CLUEWEB09 CATB DATASET

		100%	10%	9%	8%	7%	6%	5%	4%	3%	2%	1%
Postings preserved	UPP-3	1.000	0.822	0.811	0.798	0.783	0.765	0.742	0.713	0.677	0.629	0.541
	UPP-0	1.000	0.818	0.806	0.793	0.777	0.760	0.736	0.712	0.676	0.623	0.532
	UP	1.000	0.321	0.299	0.272	0.243	0.216	0.182	0.148	0.120	0.095	0.068
	TCP	1.000	0.575	0.552	0.527	0.504	0.481	0.442	0.407	0.361	0.295	0.206
Results preserved	UPP-3	1.000	0.679	0.667	0.653	0.635	0.617	0.593	0.563	0.529	0.490	0.424
	UPP-0	1.000	0.672	0.659	0.644	0.629	0.608	0.589	0.560	0.530	0.488	0.419
	UP	1.000	0.389	0.366	0.338	0.307	0.275	0.237	0.199	0.165	0.135	0.106
	TCP	1.000	0.643	0.621	0.597	0.572	0.548	0.510	0.472	0.422	0.355	0.253
P@10	UPP-3	0.276	0.266	0.264	0.262	0.258	0.257	0.257	0.251	0.243	0.236	0.214
	UPP-0	0.276	0.263	0.260	0.261	0.260	0.253	0.251	0.246	0.244	0.235	0.212
	UP	0.276	0.184	0.180	0.174	0.164	0.156	0.136	0.121	0.109	0.083	0.063
	TCP	0.276	0.277	0.274	0.275	0.273	0.272	0.266	0.257	0.245	0.223	0.203

[8] R.-C. Chen, C.-J. Lee, and W. B. Croft, "On divergence measures and static index pruning," in *Proceedings of the 2015 International Conference on The Theory of Information Retrieval*, 2015.

[9] A. Anagnostopoulos, L. Becchetti, I. Mele, S. Leonardi, and P. Sankowski, "Stochastic query covering," in *Proc. 4th ACM Int. Conf. on Web Search and Data Mining*, 2011.

[10] J. Zobel and A. Moffat, "Inverted files for text search engines," *ACM Computing Surveys*, vol. 38, no. 2, 2006.

[11] R. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*. Addision Wesley, 1999.

[12] M. Persin, J. Zobel, and R. Sacks-Davis, "Filtered document retrieval with frequency-sorted indexes," *J. of the American Society for Information Science*, vol. 47, 1996.

[13] H. Turtle and J. Flood, "Query evaluation: strategies and optimizations," *Information Processing and Management*, vol. 31, no. 6, 1995.

[14] R. Fagin, "Combining fuzzy information: an overview," 2002.

[15] X. Long and T. Suel, "Optimized query execution in large search engines with global page ordering," in *Proc. of the 29th Int. Conf. on Very Large DBs*, 2003.

[16] A. Broder, D. Carmel, M. Herscovici, A. Soffer, and J. Zien, "Efficient query evaluation using a two-level retrieval process," in *Proc. of the 12th ACM Conf. on Inf. and Knowl. Management*, 2003.

[17] T. Strohman and W. B. Croft, "Efficient document retrieval in main memory," in *Proc. of the 30th Annual Int. ACM SIGIR Conf. on Research and Development in Inf. Retrieval*, 2007.

[18] S. Ding and T. Suel, "Faster top-k document retrieval using block-max indexes," in *Proceedings of the 34th international ACM SIGIR conference on Research and development in IR 2011*, 2011.

[19] K. Chakrabarti, S. Chaudhuri, and V. Ganti, "Interval-based pruning for top-k processing over compressed lists," in *Proc. of the 27th Int. Conf. on Data Engineering*, 2011.

[20] K. Risvik, Y. Aasheim, and M. Lidal, "Multi-tier architecture for web search engines," in *1st Latin American Web Congress*, 2003.

[21] G. Leung, N. Quadrianto, A. Smola, and K. Tsioutsiouliklis, "Optimal web-scale tiering as a flow problem," in *Advances in Neural Information Processing Systems 23*, 2010.

[22] B. Cambazoglu, H. Zaragoza, O. Chapelle, J. Chen, C. Liao, Z. Zheng, and J. Degenhardt, "Early exit optimizations for additive machine learned ranking systems," in *Proc. of the ACM Int. Conf. on Web Search and Data Mining*, 2010.

[23] L. Wang, J. Lin, and D. Metzler, "A cascade ranking model for efficient ranked retrieval," in *Proc. of the 34th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval*, 2011.

[24] R. Blanco and l. Barreiro, "Static pruning of terms in inverted files," in *ECIR 2007*. Springer, 2007.

[25] S. B. Charles Clarke and P. Yeung, "Index pruning and result reranking: Effects on ad-hoc retrieval and named page finding," in *TREC 2006*.

[26] L. Nguyen, "Static index pruning for ir systems: A posting-based approach," in *Proceedings of LSDS-IR, CEUR Workshop*, 2009.

[27] E. S. de Moura, C. F. dos Santos, D. R. Fernandes, A. S. Silva, P. Calado, and M. A. Nascimento, "Improving web search efficiency via a locality based static pruning method," in *Proceedings of the 14th international conference on World Wide Web*. ACM, 2005.

[28] L. Zheng and I. J. Cox, "Entropy-based static index pruning," in *Advances in Information Retrieval*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2009.

[29] S. L. Thota and B. Carterette, "Within-document term-based index pruning with statistical hypothesis testing," in *Advances in Information Retrieval*. Springer, 2011.

[30] S. K. Vishwakarma, K. I. Lakhtaria, D. Bhatnagar, and A. K. Sharma, "An efficient approach for inverted index pruning based on document relevance," in *2014 Fourth International Conference on Communication Systems and Network Technologies*.

[31] G. Skobeltsyn, F. Junqueira, V. Plachouras, and R. Baeza-Yates, "ResIn: a combination of results caching and index pruning for high-performance web search engines," in *SIGIR conference on Research and development in information retrieval*, 2008.

[32] R. Fagin, "Combining fuzzy information from multiple systems," in *ACM SIGMOD International Conference on Management of Data and ACM Symposium on Principles of Database Systems*, 1996.

[33] Q. Wang, C. Dimopoulos, and T. Suel, "Fast first-phase candidate generation for cascading rankers," in *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2016.

[34] S. F. Chen and J. Goodman, "An empirical study of smoothing techniques for language modeling," 1996.

[35] R. Kneser and H. Ney, "Improved backing-off for m-gram language modeling," 1995.

[36] P. Nguyen, J. Gao, and M. Mahajan, "MSRLM: a scalable language modeling toolkit," Microsoft Research, Tech. Rep., 2007.