Static Index Pruning for Information Retrieval Systems

David Carmel*, Doron Cohen*, Ronald Fagin§, Eitan Farchi*,
Michael Herscovici*, Yoëlle S. Maarek*, Aya Soffer*

(*)IBM Research Lab in Haifa, MATAM, Haifa 31905, ISRAEL
(§)IBM Almaden Research Center, 650 Harry Road, San Jose, CA 95120

ABSTRACT

We introduce *static index pruning* methods that significantly reduce the index size in information retrieval systems. We investigate *uniform* and *term-based* methods that each remove selected entries from the index and yet have only a minor effect on retrieval results. In uniform pruning, there is a fixed cutoff threshold, and all index entries whose contribution to relevance scores is bounded above by a given threshold are removed from the index. In term-based pruning, the cutoff threshold is determined for each term, and thus may vary from term to term. We give experimental evidence that for each level of compression, term-based pruning outperforms uniform pruning, under various measures of precision. We present theoretical and experimental evidence that under our term-based pruning scheme, it is possible to prune the index greatly and still get retrieval results that are almost as good as those based on the full index.

Topic areas: indexing, compression

1. INTRODUCTION

Fast and precise text search engines are widely used in Web and desktop applications. Efficient query evaluation is attained in these search engines by use of an inverted file, which provides an association between terms[1] and documents in the collection. Indexing a large collection of documents might result in huge index files that are hard to maintain. Therefore, it is important to utilize efficient compression methods for index files.

While many index compression schemes have been suggested in the past, two recent trends lead us to believe that the issue of index size needs to be revisited. The first trend is the emergence of hand-held devices such as Palm that now possess enough storage capacity to allow moderate-size document collections to be stored on the device. Equipping these devices with advanced index-based search capabilities is desirable, for quick reference and browsing purposes, but storage on hand-held devices is still sufficiently limited that indices need to be very compact. The second trend is the explosion of textual information on the Web; thus, Web search engines must index collections of unprecedented size.

There has been a large amount of work in the field of index compression. There are two complementary approaches: *lossless compression* and *lossy compression*. Lossless approaches do not lose any information; instead, they use more efficient data structures. Thus, under lossless approaches, posting lists have a very compact representation [16]. On the other hand, under lossy approaches, certain information is discarded. The two approaches are complementary. That is, after selected document postings have been pruned using lossy methods, the index can be further compressed in a lossless manner. Thereby, a smaller index size can be attained than is possible by either one of the methods separately.

Examples of the lossy approach include stopword omission and Latent Semantic Indexing (LSI) [3]. While the primary goal of both stopword omission and LSI is to reduce the noise in the system by pruning terms that deteriorate precision, their practical effect of reducing index size is very relevant to this work. Both stopword omission and LSI operate at the granularity of terms. That is, they enable pruning only of an entire term and its postings list from the index.

In this paper we propose lossy methods that prune the index at the posting level. That is, in our approach, a term can be retained in the index, but some document postings may be eliminated from this term's posting list. The idea is to remove those postings whose potential contribution to the relevance score of a document is so small that their removal will have little effect on the accuracy of the system. The selection of which document postings to prune is guided by certain user-specified parameters.

In addition to reducing disk space, our pruning methods decrease both the time required to search the resulting index, and the amount of memory consumed for searching, in a manner similar to that of dynamic pruning techniques that have been described in the literature. These techniques dynamically decide, during query evaluation, whether certain terms or document postings are worth adding to the accumulated document scores, and whether the ranking process should continue or stop [1, 6, 9, 14]. The typical approach of these algorithms is to order query terms by decreasing weight, and to process terms from most to least significant until some stopping condition is met. Among the stopping conditions are a threshold on the number of document accumulators used, and a threshold on term frequencies. While these schemes select terms, and hence whole posting lists,

[1]We use "term" to refer to an arbitrary indexing unit including a single word, a phrase, or a more complex form.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for pro£t or commercial advantage and that copies bear this notice and the full citation on the £rst page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior speci£c permission and/or a fee.
SIGIR'01, September 9-12, 2001, New Orleans, Louisiana, USA..
Copyright 2001 ACM 1-58113-331-6/01/0009 ...$5.00.

for processing or rejection, Persin et al. [10, 11] propose a method that prunes entries within these lists. In this approach, the processing of a particular query term is terminated when a stopping condition is met. For an overview of these and other query evaluation methods, see [13].

By contrast, our pruning methods are static. We remove entries from the index in advance, therefore reducing the index size. Similarly to lossy versus lossless compression, the two different techniques of static versus dynamic pruning can complement each other, since searching in the pruned index can be further accelerated by dynamic pruning for a specific query.

Our goal is to perform index pruning in such a way that a human "cannot distinguish the difference" between the results of a search engine whose index is pruned and one whose index is not pruned. Therefore, as in any lossy compression technique, we wish to remove the least important entries from the index, so that the visible effects of the compression (in terms of the results obtained) are very small. Thus, the question we need to address is how to identify the least important entries in the index. We begin with the usual assumption that for each query, there is a *scoring function* that assigns a score to each document, so that the documents with the highest scores are the most relevant. The scoring function is often based on a 2-dimensional *scoring table* A, indexed by terms and documents. Table entries are set according to the scoring model of the search engine; thus, $A(t, d)$ is the score of document d for term t. We assume that $A(t, d) = 0$ if t is not in d, and $A(t, d) > 0$ otherwise. There are no other constraints on table entries.

The first static pruning algorithm that we consider removes from the index all posting entries whose corresponding table values are bounded above by some fixed cutoff threshold. We refer to this type of pruning as *uniform pruning*, since the threshold is uniformly chosen, with the same cutoff value being used for every term. Uniform pruning has an obvious drawback. Low-scoring terms may have all of their entries in the index pruned away. Therefore, given a query consisting only of low-scoring terms, the pruned index may fail to provide any good results for this query.

This insight leads us to suggest a second, and more sophisticated, pruning algorithm, in which the cutoff threshold may depend on the term. We refer to this type of pruning as *term-based pruning*. Term-based pruning guarantees that each term will have some representatives left in the index. Therefore, queries with low-scoring terms will fare better than under uniform pruning. How do we determine the cutoff thresholds?

We are guided by the intuition that all we care about are the top k documents (say, for $k = 10$ or $k = 50$), since this is all the user sees. Thus, we care only about whether the pruned index returns the same top k documents; we do not care about the score it might assign to the remaining documents. Our term-based pruning algorithm attempts to minimize the effect of pruning on the top k results for each query (k is one of the parameters of the algorithm). We now discuss the ideas that go into term-based pruning.

We define an *ϵ-variation* of a scoring function S to be a scoring function S' that differs from S by at most a multiplicative factor of ϵ. The intuition is that scoring functions are inherently fuzzy, and so S' is essentially as good as S. We present an idealized term-based pruning algorithm, which selects a cutoff threshold for each term, such that the scoring function from the pruned index has the same top k answers, in the same order, as some ϵ-variation of the original scoring function. This corresponds to our intuitive notion of saying that a human "cannot distinguish the difference". Thus, as we will show, the idealized term-based pruning algorithm has a mathematical guarantee that the top k answers are good, in a certain precise sense.

Because of its mathematical guarantees, we would like to use this idealized term-based pruning algorithm to set the cutoff thresholds. Unfortunately, as we shall discuss, there are practical reasons why it does not do sufficient pruning. Therefore, we instead define another term-based pruning algorithm, whose cutoff thresholds for each term are based on those of the idealized term-based pruning algorithm. We show by simulations that we are able to achieve 35% pruning with only a slight decrease in average precision (7%), and with hardly any effect on the precision of the top 10 results. For 50% pruning, we are still able to maintain the same precision at the top 10 results.

In Section 2 we define precisely the model used by our algorithms and the notion of query result indistinguishability. In Section 3, we give (two variations of) the idealized term-based pruning algorithm. We give a formal proof that for all queries with a moderate number of search terms, the scoring function obtained from the pruned index using the idealized term-based algorithm is indistinguishable from that obtained from the original index. In Section 4 we tell how we modify the idealized term-based pruning algorithm in practice, and we show experimentally, using TREC data, that our (modified) term-based pruning algorithms attain a very high degree of pruning. While uniform pruning adversely affects the average precision, term-based pruning results in a pruned index that is almost as good as the original index in terms of recall-precision evaluation and in terms of similarity between the top k search results. Thereby, under term-based pruning, we obtain a greatly compressed index that gives answers that are essentially as good as those derived from the full index. In Section 5, we give our conclusions.

2. DEFINITIONS AND NOTATION

A scoring function S is a function mapping each document to a nonnegative real number. Often, S is derived from a scoring table A, as defined in the introduction, and a query, as we now explain.

A typical scoring model used by many IR systems is the $tf \times idf$ formula [12]. The score of term t for document d depends on the term frequency tf of t in d, the length $|d|$ of document d, and the inverse number idf of documents containing t in the collection. For example, our system uses the following scoring model based on the model used by the SMART system [2]

$$A(t, d) = \frac{\frac{\log(1+tf)}{\log(1+avgtf)} \log \frac{N}{N_t}}{|d|} \tag{1}$$

where $avgtf$ is the average term frequency in d, N is the number of documents in the collection, N_t is the number of documents containing t, and

$$|d| = [0.8 * \text{pivot} + 0.2 * (\# \text{ unique terms in d})]^{0.5}$$

is an approximation to the length of document d. The pivot is set to the average number of unique terms in a document occurring in the collection.

Note that since (a) search indices are typically very large and contain many entries, and (b) the size of the index should be as small as possible, IR systems usually store the raw term frequencies (which are integers) rather than the actual scores (which are floating point numbers). The table entries are computed dynamically according to the scoring model of the system during query evaluation time. Furthermore, only nonzero entries are kept in the index. In particular, the index is stored using an inverted file structure that consists of terms, where each term is associated with a posting list that records the documents that contain the term and the number of occurrences of the term in the document (and perhaps more information, such as the offsets of the term in the document).

We identify a *query* q with a nonempty set of distinct terms $t_1, \ldots, t_r$, where each term t_i is associated with a positive weight α_i. For example, in the experiments done for this work, query terms are extracted from TREC topics, and the term weights are determined by the following equation (also based on the scoring model used by the SMART system [2]):

$$\alpha_i = \frac{\log(1 + tf_i)}{\log(1 + avgtf)} \tag{2}$$

Here tf_i is the term frequency of term t_i in the topic and *avgtf* is the average term frequency in the topic.

The score of document d for query q is

$$S_q(d) = \sum_{i=1}^{r} \alpha_i A(t_i, d). \tag{3}$$

We say that S_q is the scoring function for q that is *based on* A.

Assume that $0 < \epsilon < 1$. Let us say that a scoring function S' is an *ϵ-variation* of the scoring function S if

$$(1 - \epsilon)S(d) \leq S'(d) \leq (1 + \epsilon)S(d) \tag{4}$$

for each document d. We think of ϵ as representing the fuzziness inherent in the scores. Under this interpretation, if S' is an ϵ-variation of S, then intuitively, the score of document d could just as well be $S'(d)$ as $S(d)$.

Let A be a scoring table. Our goal is to find a new scoring table A^* that is like A, except that some of the entries are set to 0. Using the uniform pruning approach, we find a uniform cutoff threshold τ for the entire table such that $A^*(t, d) = A(t, d)$ if $A(t, d) > \tau$, and $A^*(t, d) = 0$ otherwise. Using the term-based pruning approach, for each term t, we find a cutoff threshold τ_t such that $A^*(t, d) = A(t, d)$ if $A(t, d) > \tau_t$, and $A^*(t, d) = 0$ otherwise. Thus, A^* is the result of "cutting the tail" of A (since we are removing the smallest entries from the table, in the uniform case, and from each row, in the term-based case).

In term-based pruning, we wish to cut the tail in a principled manner. Intuitively, our principle is that for each query q (or at least for each query q without too many terms), the resulting scoring function based on A^* should still be acceptable. We now explain what we mean by "acceptable".

In information retrieval, we are most interested in the "top answers" to a query. We consider two notions of the "top answers" to a query q, as follows.

- **Top k answers**, where k is a positive integer: these are the k documents with the highest scores under the query q (ties are broken arbitrarily).
- **δ-top answers**, where $0 < \delta \leq 1$: these are the documents whose score under the query q is at least δ times the highest score of all the documents under q. For example, for $\delta = 0.7$, each document with a score that is at least 70% of the top score is a δ-top answer.

The top k answers definition is more intuitive, since search results are usually presented k (e.g., 10) at a time, and users often want to see only the top k, and no more. However, if the quality of the top results is of significance, then showing an arbitrary and preset number of results may fail to convey such qualitative information to the user. Perhaps only the first three results are good; or, on the other hand, perhaps there are a few more results that are just as good as the tenth result. Using the δ-top definition, all the results that are presented to the user can be considered significant. While for one query there may be only three documents with scores that are significantly higher than others, for another query there may be twenty such documents.

We define S_q^* to be the scoring function for query q based on the scoring table A^*, where A^* is the result of cutting the tail of our original scoring table A. We consider S_q^* to be acceptable if it gives the same top answers (in the same order) as some ϵ-variation of S_q. The intuition is that what the user really cares about is the top answers. Since there is an inherent fuzziness in the scoring functions, giving the same top answers as an ϵ-variation S' of S_q is good enough. The reason we need both S_q^* and S' is that S_q^* itself is not necessarily an ϵ-variation of S_q, since the result of cutting the tail might result in $S_q^*(d)$ being equal to 0 for some document d, whereas $S_q(d)$ is nonzero. This keeps S_q^* from being an ϵ-variation of S_q.

Let us say that the scoring function S_q^* is *(k, ϵ)-good* for S_q if there is an ϵ-variation S' of S_q such that the top k answers for S_q^* are the same as the top k answers for S', in the same order. Similarly, S_q^* is *(δ, ϵ)-good* for S_q if there is an ϵ-variation S' of S_q such that the δ-top answers for S_q^* are the same as the δ-top answers for S', in the same order. We will give theorems that say that our idealized term-based pruning algorithms attain this type of goodness.

3. IDEALIZED TERM-BASED PRUNING

In this section we give two variations of our idealized term-based pruning algorithm. The first corresponds to the "top k answers" definition, while the second corresponds to the "δ-top answers" definition. For both of these algorithms, we provide mathematical guarantees of goodness.

3.1 Idealized Top k Answers Pruning Algorithm

Given the parameters ϵ and k, we shall give a method for doing the pruning (that is, for obtaining table A^* from table A) that guarantees that for all queries q with r terms, such that $r < 1/\epsilon$, the scoring function S_q^* based on A^* is $(k, \epsilon r)$-good for S_q. Note that r is limited by the inequality $r < 1/\epsilon$, since the notion "$(k, \epsilon r)$-good" is meaningful only for $\epsilon r < 1$. If we wish to guarantee that for all queries q with at most ℓ terms, the scoring function S_q^* based on A^* is (k, ϵ)-good, then we simply modify our construction by using ϵ/ℓ instead of ϵ. Although the method guarantees good results only for short queries (those with r not too large), the pruned index can apparently be successfully used for longer queries as well. Thus, in Section 4, we show experimentally that a

practical variation of our idealized term-based pruning algorithms gives good results even for longer queries.

We now give our construction, which tells how we prune the scoring table A to obtain A^*, given our parameters ϵ and k. For each term t, let z_t be the score of the kth highest ("kth best") document for term t according to A. Let $\tau_t = z_t\epsilon$. We take τ_t to be the cutoff threshold for term t. Thus, let $A^*(t,d) = A(t,d)$ if $A(t,d) > \tau_t$, and $A^*(t,d) = 0$ otherwise. In other words, the rule for cutting the tail is to set to 0 those scores that are at most τ_t.

Recall that as we explained in Section 2, the scoring table A is not stored as such in typical IR systems. Instead, we use an inverted file, where each term is stored with an associated posting list. Figure 1 describes how to prune a given inverted file using the top k pruning algorithm. The algorithm takes as input an inverted file I, along with the parameters k and ϵ, and creates a pruned inverted file. Note that the entries of the scoring table A are computed on a term-by-term basis, in order to find the cutoff value for each particular posting list.[2] The time complexity of the pruning

```
top k prune(I, k, ε)
  for each term t in I
    retrieve the posting list P_t from I
    if |P_t| > k
      for each entry d ∈ P_t
        compute A(t, d) (e.g. by Equation 1)
      let z_t be the kth best entry in row t of A
      τ_t ← ε · z_t
      for each entry d ∈ P_t
        if A(t, d) ≤ τ_t
          remove entry d from P_t
    Save (t, P_t) in the pruned inverted file
```

Figure 1: Idealized top k pruning algorithm

algorithm is linearly proportional to the index size. For each term t, the algorithm first computes a threshold by finding the kth best entry in the posting list of t (this can be done in $O(N)$ time, where N is the number of documents). It then scans the posting list to prune all the entries smaller than the threshold. Thus, if there are M terms in the index, the time complexity of the algorithm is $O(M \cdot N)$.

Before we state and prove our theorem, we need to define carefully what we mean by "the top k answers". This is a little delicate since there may be ties. Given a scoring function S, a set X is said to be the top k answers if X is of size k, and if for every $x \in X$ and every $y \notin X$ we have $S(x) \geq S(y)$. Because of ties, there may be more than one set X that can serve as the top k answers for a given query. In the definition of (k,ϵ)-good (and of (δ,ϵ)-good), we assume that ties are broken in the same way in S_q^* as in S'.

Assume that the scoring table A and the parameters ϵ and k are given. Let A^* be the pruned scoring table obtained by our construction described earlier. We then have the following theorem.

[2]The scoring table derived from the pruned posting lists is actually slightly different from the pruned scoring table defined earlier where certain entries are set to zero and other entries are unchanged. This is because the *idf* values in the *tf* $\times$ *idf* formula we use change slightly after pruning the posting lists. For simplicity, we ignore this issue here.

THEOREM 3.1.: *Let q be a query with r terms, where $r < 1/\epsilon$. Let S_q be the scoring function for query q based on the scoring table A, and let S_q^* be the scoring function for query q based on the pruned table A^*. Then S_q^* is $(k,\epsilon r)$-good for S_q.*

Proof: Let X be the top k answers for S_q^*. Thus, X is a set of size k, and for every $x \in X$ and every $y \notin X$ we have $S_q^*(x) \geq S_q^*(y)$. Define S' by taking $S'(x) = S_q^*(x)$ if $x \in X$, and $S'(y) = (1-\epsilon r)S_q(y)$ if $y \notin X$. We now show that S' is an ϵr-variation of S_q. In fact, we show something a little stronger, since instead of just the inequalities $(1-\epsilon r)S_q(d) \leq S'(d) \leq (1+\epsilon r)S_q(d)$, we prove the stronger statement $(1-\epsilon r)S_q(d) \leq S'(d) \leq S_q(d)$. The second inequality $S'(d) \leq S_q(d)$ follows easily from the construction. We now prove the first inequality. It is immediate if $d \notin X$. Assume now that $x \in X$; we must show that

$$(1-\epsilon r)S_q(x) \leq S'(x). \tag{5}$$

Assume that the terms of the query q are $t_1, \ldots, t_r$, with weights $\alpha_1, \ldots, \alpha_r$ respectively. Let z_i be the score of the kth highest answer for term t_i according to A, for $1 \leq i \leq r$. Find j (with $1 \leq j \leq r$) such that $\alpha_j z_j$ is as big as possible. Let W be the top k answers according to A for term t_j. It follows from our construction that for each $w \in W$, we have $A^*(t_j, w) = A(t_j, w) \geq z_j$, and so $S_q^*(w) \geq \alpha_j z_j$. Therefore, there are at least k documents w (namely, the members of W) such that $S_q^*(w) \geq \alpha_j z_j$. Since x is one of the top k answers for S_q^*, it follows that

$$S_q^*(x) \geq \alpha_j z_j. \tag{6}$$

Since $S_q(x) \geq S_q^*(x)$, it follows from (6) that $S_q(x) \geq \alpha_j z_j$. Therefore,

$$\begin{aligned} S'(x) &= S_q^*(x) \\ &\geq S_q(x) - \sum_{i=1}^{r} \alpha_i z_i \epsilon \\ &\quad \text{(since the sum is the maximal loss from pruning)} \\ &\geq S_q(x) - r\alpha_j z_j \epsilon \text{ (since } \alpha_j z_j \geq \alpha_i z_i \text{ for all } i) \qquad (7) \\ &\geq S_q(x) - rS_q(x)\epsilon \text{ (since } S_q(x) \geq \alpha_j z_j) \\ &= (1-\epsilon r)S_q(x). \end{aligned}$$

This proves (5).

Assume that $x \in X$ and $y \notin X$. We shall show that

$$S'(x) \geq S'(y). \tag{8}$$

There are two cases, depending on whether $\alpha_j z_j \geq S_q(y)$ or $S_q(y) > \alpha_j z_j$.

Case 1: $\alpha_j z_j \geq S_q(y)$. By (6) it follows that $S_q^*(x) \geq S_q(y)$. Hence, $S'(x) = S_q^*(x) \geq S_q(y) \geq S'(y)$. This proves (8), as desired.

Case 2: $S_q(y) > \alpha_j z_j$. Within (7) we showed $S_q^*(x) \geq (1-\epsilon r)S_q(x)$. Since $S_q(y) > \alpha_j z_j$, the same argument (with x replaced by y) shows that $S_q^*(y) \geq (1-\epsilon r)S_q(y)$. Hence, $S'(x) = S_q^*(x) \geq S_q^*(y) \geq (1-\epsilon r)S_q(y) = S'(y)$. This proves (8), as desired.

Thus, X is also the top k answers for S'. Since S' is an ϵr-variation of S_q and since S' and S_q^* agree on X, it follows easily that S_q^* is $(k,\epsilon r)$-good for S_q , which proves the theorem. ∎

An examination of the proof shows that Theorem 3.1 can be strengthened in two ways. First, in addition to S_q^* having

the same top k answers, in the same order, as an ϵr-variation S' of S_q, also these top k answers have the same score under S_q^* as S'. Therefore, if the user were to see not only the top k, but also the scores of the top k, then what the user would see using S_q^* would be identical to what he would see using S'.

Second, nowhere in the proof of Theorem 3.1 did we make use of the fact that certain entries of A are set to 0 (that is, we never made use of the fact that some entries are set to 0, only that the entries that are set to 0 are sufficiently small). Therefore, if our algorithm were modified by not setting to 0 some entry that the algorithm says to set to 0, then the results of Theorem 3.1 would still hold.

3.2 Idealized δ-Top Answers Pruning Algorithm

Given the parameters ϵ and δ, we shall give a method for doing the pruning (that is, for obtaining table A^* from table A) that guarantees that for all queries q with r terms such that $r < 1/\epsilon$, the scoring function S_q^* based on A^* is $(\delta, \epsilon r)$-good for the scoring function S_q based on A.

We now give our construction, which describes how we prune the scoring table A to obtain A^*, given our parameters ϵ and δ. For each term t, let $z_t = \delta \max_d A(t,d)$ be the δ-top score for term t according to A. We take $\tau_t = \epsilon z_t$ to be the cutoff threshold for term t. Thus, let $A^*(t,d) = A(t,d)$ if $A(t,d) > \tau_t$, and $A^*(t,d) = 0$ otherwise. Similarly to the top k algorithm, the time complexity of the δ-top algorithm is $O(M \cdot N)$.

Assume that the scoring table A and the parameters ϵ and δ are given. Let A^* be the pruned scoring table obtained by our construction described earlier. We then have the following theorem, whose proof is very similar to that of Theorem 3.1.

THEOREM 3.2.: *Let q be a query with r terms, where $r < 1/\epsilon$. Let S_q be the scoring function for query q based on the scoring table A, and let S_q^* be the scoring function for query q that is based on the pruned table A^*. Then S_q^* is $(\delta, \epsilon r)$-good for S_q.*

4. EXPERIMENTAL RESULTS

In this section we tell how we modified the idealized term-based pruning algorithms based on practical considerations. We report results from some experiments we conducted to evaluate how the pruning algorithms behave, both for short queries (where our theory applies) and for long queries. For our experiments we use Juru, a Java version of the Guru [8] search engine, developed at the IBM Research Lab in Haifa. Juru does not apply any compression methods on its inverted files, except stopword omission, and uses the $tf \times idf$ formula described in Equation 1. We use the TREC collection [15] for our experiments. We used the Los-Angeles Times (LAT) data given in TREC, which contains about 132,000 documents (476MB), and evaluated our methods on the ad hoc tasks for TREC-7, with topics 401-450. We experimented both with short queries and long queries. We used the topic title for short query construction, and the title concatenated with the topic description for long query construction. The weights of the query terms were determined by Equation 2.

4.1 Performance Measurement

In order to measure the accuracy of the pruning algorithms, we compare the results obtained from the pruned index against the results obtained from the original index. We use the standard average precision, and precision at k ($P@k$) measures [15].

In addition, we want a method to compare the top results in one list (obtained from the pruned index) against the top results in another list (obtained from the original index). In this way we can provide numbers saying how well the pruned index approximates the original index in terms of the top results. A simple measure is the symmetric difference between the top k lists, which evaluates how similar the lists are, ignoring the order of results. If y is the size of the union of the two top k lists, and x is the size of the symmetric difference, then we take the symmetric difference score to be $1 - \frac{x}{y}$. This score lies between 0 and 1. The highest score of 1 occurs precisely when the top k of both lists are the same (although the order of results may be different), and the lowest score of 0 occurs precisely when the top k of the two lists are disjoint.

In order to take the document order into consideration there is a need for a new similarity measure. Recently, Fagin et al. [4], inspired by the question of measuring the goodness of our pruning approach, developed various correlation methods for comparing the top k in two lists, based on methods for comparing two permutations [7]. For our experiments, we used one of their variations of Kendall's tau method. The original Kendall's tau method for comparing two permutations assigns a penalty $S(i,j) = 1$ for each pair $\{i,j\}$ of distinct items for which i appears before j in one permutation and j appears before i in the other permutation. The sum of the penalties over all pairs $\{i,j\}$ reflects an overall penalty score that lies between 0 (when permutations are identical) and $n(n-1)/2$ (when one permutation is the inverse of the other).

The modified version of Kendall's tau handles the case where we care about comparing the top k in one list against the top k in another list, rather than comparing permutations. The penalties assigned for each pair $\{i,j\}$ of distinct items needs to be redefined, since i and j might not appear in the top k of one or both lists. For each i,j, each of which appear in the top k of at least one of the lists, the new penalty $S(i,j)$ is defined as follows:

- Case 1: if i and j appear in the top k of both lists, then $S(i,j)$ is defined as before. Namely, if i and j are in the same order, then $S(i,j) = 0$, and otherwise $S(i,j) = 1$.
- Case 2: if i and j appear in the top k of one list (say list 1), and exactly one of i or j (say i), appears in the top k of the other list (list 2), and if i is ahead of j in list 1, then $S(i,j) = 0$, and otherwise $S(i,j) = 1$. Intuitively, we know that i is actually ahead of j in list 2, since i, but not j, appears in the top k of list 2.
- Case 3: if i, but not j, appears in the top k of one list, and j, but not i, appears in the top k of the other list, then $S(i,j) = 1$. Intuitively, we know that i is actually ahead of j in the one list, and j is actually ahead of i in the other list.
- Case 4: if i and j appear in the top k of one list, but neither i nor j appear in the top k of the other list,

then $S(i,j) = 1/2$. Intuitively, we do not have enough information to determine whether i and j are in the same order in list 1 as in list 2, and so we assign a neutral penalty score of 1/2.

The overall penalty score now lies between 0 (when the top k of the lists are identical and in the same order) and $k(3k-1)/2$ if the top k of one list is disjoint from the top k of the other list. We get a normalized overall penalty score x', which lies between 0 and 1, by setting $x' = 1 - \frac{2x}{k(3k-1)}$. The highest score of 1 occurs precisely when the top k of both lists are the same, in the same order, and the lowest score of 0 occurs precisely when the top k of the two lists are disjoint.

4.2 Results

The first experiment tested the impact of pruning on (a) the search precision and (b) the similarity of the top results obtained from the original index to the top results obtained from the pruned index. First, we created a sequence of pruned indices using the uniform pruning algorithm, where we varied τ, the cutoff threshold, and thereby the number of entries pruned from the resulting pruned index. Next, we created a sequence of pruned indices by invoking the top k pruning algorithm, where we fixed k to 10 and where we used varying values of ϵ. We conducted a similar sequence of experiments by invoking the δ-top algorithm, where we fixed δ to 0.7 and used varying values of ϵ. For each index we ran a set of 50 short queries and a set of 50 long queries, constructed from the TREC topics, and measured the precision of the search results and the similarity of the top 10 results to the top 10 results of the original index. The average number of terms in the short queries (after stemming and stopword elimination) is 2.42. The average number of terms in the long queries is 9.06.

We now explain why we were required to modify the idealized term-based pruning algorithms. The initial results using the idealized term-based algorithms were discouraging. Setting the cutoff threshold to the kth highest value of each term times small ϵ values did not prune a significant number of entries in the index. Careful examination of the values of table A revealed that the problem was the small range of values in the table derived from our scoring function. In particular, the value of the smallest entry in each row was almost always larger than the cutoff threshold. We thus decided to shift the nonzero values in the table A so that the lower limit of the range of values is close to 0. Shifting is done by subtracting the smallest positive table entry from all positive table entries. We then applied the idealized term-based pruning algorithm, in order to determine which documents should be removed from various posting lists. This shift indeed resulted in significant pruning. The net result was a pruned index with little deterioration in precision and with hardly any change in the top k results. While the theorems presented in Section 3 do not hold for the shifted table, the experimental results show that the modified term-based algorithm is able to prune a significant portion of the index and still yield results that are essentially as good as the original index. We have obtained preliminary results that give some partial theoretical justification for the modified algorithm, but further research is required to fully understand the phenomenon.

The first results we present compare the uniform and term-based pruning methods both in terms of precision and in terms of the similarity between the top 10 lists for various levels of pruning. The level of pruning is determined by τ in the uniform algorithm, and by ϵ in the term-based approaches. We then analyze the term-based algorithms for short and long queries as we vary ϵ. Finally, we examine the effect of varying k and δ for fixed ϵ.

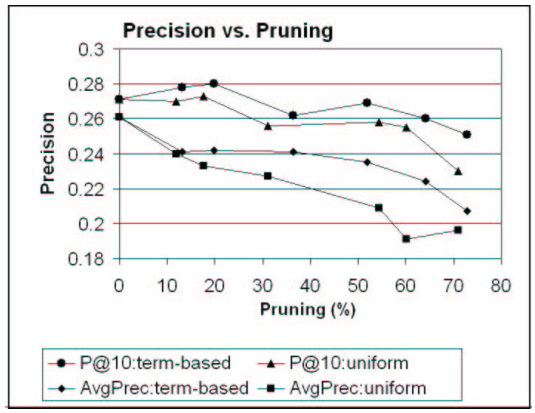

Figure 2: Precision of search results at varying levels of pruning.

Figure 2 plots the average precision and the $P@10$, as a function of the amount of pruning, for the indices obtained by the uniform and the top k term-based algorithms. The results show that the top k term-based algorithm is overall better than the uniform algorithm. They also indicate that while there is a significant deterioration in the average precision with the increase of pruning, there is only moderate deterioration in the precision at the top 10 results.

Note that the pruning reported in our results is the percentage of entries in the index that are removed by our algorithms. The effect of this pruning on the index size depends on the particular data structures used to store the index. In our system, the saving in space is linearly proportional to the savings in terms of index entries. As mentioned above, we did not employ any lossless compression methods, since we wanted to study the effects of static pruning in isolation of other factors. Clearly, the total size of the index can be further compressed by applying lossless compression to the pruned index. It may very well be the case that the overall savings in the compressed space is less than that reported here, since lossless compression of a smaller index might not lead to as much compression as lossless compression of a larger index. Investigating how static pruning and various traditional compression methods interact is an interesting topic for research and left for future work.

Figure 3 shows the similarity between the top 10 results of the original index and the pruned index, at varying levels of pruning, both for uniform pruning and top k term-based pruning (with $k = 10$). We use the symmetric difference and the top k-based version of Kendall's tau to measure the similarity between the two lists. The results are averaged over the 50 long queries. This graph also shows the superiority of the top k term-based algorithm over the uniform algorithm. The relatively high similarity between the top 10 lists for moderate pruning levels supports the claim that for moderate pruning levels, the top 10 results of the pruned indices are very similar to the top 10 results of the original index.

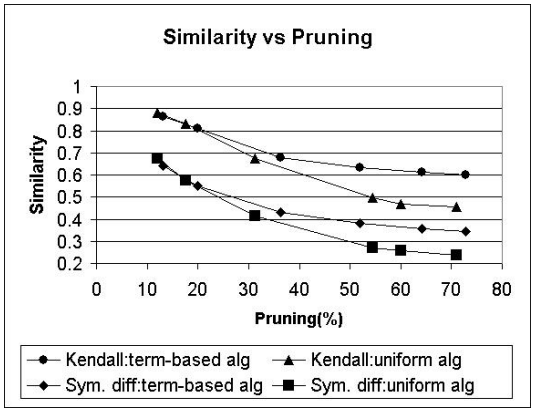

Figure 3: Top 10 similarity at varying levels of pruning.

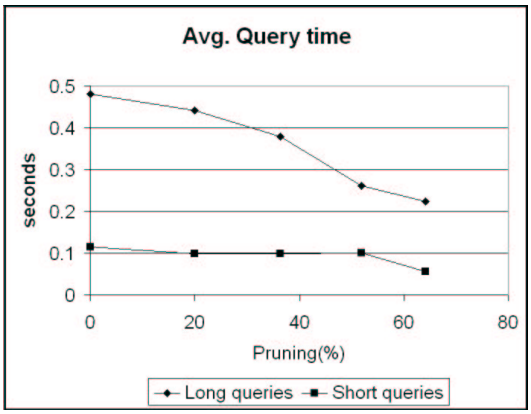

Figure 4: Average query processing time at varying levels of pruning.

Figure 4 shows the average query processing time for long and short queries, as a function of the amount of pruning. The results clearly show that especially for long queries, query time processing is significantly lower for smaller indices. Note that we did not employ any dynamic pruning methods in our experiments. These results thus show that static pruning in itself yields shorter query processing time. Query processing can be made even more efficient by employing additional dynamic pruning methods.

ϵ	Prune(%)	Avg precision		P@10	
		LongQ	ShortQ	LongQ	ShortQ
0	0	0.261	0.250	0.271	0.262
0.025	13.2	0.241	0.242	0.278	0.280
0.05	19.9	0.242	0.240	0.280	0.282
0.1	36.4	0.241	0.242	0.262	0.278
0.15	51.9	0.226	0.235	0.269	0.258
0.2	64.2	0.224	0.229	0.260	0.256
0.25	72.8	0.207	0.223	0.251	0.249

Table 1: Precision results of the top k term-based algorithm for $k = 10$ and varying ϵ.

Table 1 summarizes the results obtained by the top k term-based algorithm, for long and short queries, where k is fixed to be 10 and where ϵ is varied. As the results show, the algorithm can achieve high levels of pruning even for modest values of ϵ. For example, there is a pruning level of 72.8% even for $\epsilon = 0.25$. Note that while the average precision decreases with the pruning level, the precision at the top 10 results is hardly effected, and even improves for small levels of pruning. Surprisingly, there is very little difference in precision of the pruned indices for short queries versus long queries.

ϵ	Prune(%)	Avg precision		P@0.7-top	
		LongQ	ShortQ	LongQ	ShortQ
0	0	0.261	0.250	0.330	0.320
0.05	19.4	0.239	0.240	0.329	0.310
0.1	35.6	0.236	0.236	0.314	0.311
0.15	51.4	0.220	0.223	0.323	0.294
0.2	64.4	0.228	0.231	0.331	0.304

Table 2: Precision results of the δ-top term-based algorithm for $\delta = 0.7$ and varying ϵ.

Table 2 summarizes the results obtained by the δ-top algorithm, fixing δ to 0.7 and varying ϵ. The average precision of the search results are consistent with the results obtained from the top k algorithm. The fourth and the fifth columns show the precision at the 0.7-top result lists obtained from the pruned indices (all results higher than 0.7 times the top result). Note that the precision at the 0.7-top results is much higher than the precision at the top 10 results. The main reason for this is that the documents in the δ-top list, by definition, have a significantly higher score than the scores of the rest of the documents. Thus the documents that are tested for relevancy to the query are more likely relevant.

k	Prune(%)	Avg precision	P@k	
			Orig Index	Pruned Index
1	49.2	0.218	0.533	0.356
5	40.2	0.231	0.347	0.293
10	36.4	0.241	0.271	0.262
15	34.2	0.238	0.241	0.236

Table 3: Precision results of the top k algorithm for $\epsilon = 0.1$ and varying k.

Table 3 summarizes the results obtained by the top k algorithm for long queries, where ϵ is fixed to 0.1 and k is varied. For smaller values of k, the algorithm prunes much more, as expected, since the cutoff threshold is higher. As k increases, the precision of the top k results of both the original and the pruned indices decreases. Examining the difference between precision of the original and pruned indices shows that the adverse effect of pruning on precision becomes less significant with increase in k, since larger values of k result in less pruning.

δ	Prune(%)	Avg precision	P@δ-top	
			Orig Index	Pruned Index
0.99	48.9	0.217	0.516	0.456
0.9	44.9	0.230	0.468	0.410
0.8	40.4	0.234	0.430	0.370
0.7	35.6	0.236	0.330	0.314

Table 4: Precision results of the δ-top algorithm for $\epsilon = 0.1$ and varying δ.

Table 4 summarizes the results obtained by the δ-top algorithm for long queries, where ϵ is fixed to 0.1 and δ is varied.

The size of the δ-top set is varied with δ. For $\delta = 0.99$ the average size of the 0.99-top set is 1, while for $\delta = 0.7$ the average set size is 16.9.

5. CONCLUDING REMARKS

In this work we present static pruning methods that reduce the index size of a search engine by removing the least important posting entries from the index. We consider two types of static pruning. *Uniform pruning* removes all posting elements whose values are bounded above by some fixed cutoff threshold. *Term-based pruning* proceeds similarly, but assigns a possibly different cutoff threshold to each term in the index. We present an idealized term-based pruning algorithm, and prove that for short queries the scoring function based on the pruned index has the same top answers, in the same order, as some ϵ-variation of the original scoring function. Thus, the top answers returned by the pruned index are guaranteed to be "good" in a certain precise sense.

Because of practical reasons, the idealized term-based pruning algorithm does not prune much, at least in our experiments. Therefore we modify the idealized term-based algorithm into a practical term-based algorithm whose cutoff thresholds are based on those of the idealized one. We show by simulations that we can achieve high levels of pruning, while still retrieving "good" results in terms of (a) precision and (b) similarity to the top results of the original unpruned index. Therefore, this work essentially shows how to obtain a greatly compressed index that still gives answers that are almost as good as those derived from the full index.

There are many interesting open questions. The first one is whether our lossy methods are effective for small indices. We intend to experiment with small indices using *Pirate*, a compact search engine that we have developed for Palm. (*Pirate* is available as a free download at [5].) The goal is to attain an index that is very small, which is required by this device because of its limited storage. We also intend to index and prune the Web Track of TREC ($\sim$10G) in order to experiment with huge indices. There are theoretical reasons to believe that our term-based algorithms will be even more effective in the case of huge indices, and we wish to investigate this issue empirically. In addition to these empirical studies, there are further theoretical issues that we wish to explore. In particular, we would like to find a theoretical justification for our practical modification to idealized term-based pruning, to complement the strong empirical evidence we have presented here. We do have some initial weak theoretical results in this case that show some form of "closeness" between the results obtained from the pruned index and the original index. In those results, we greatly liberalize the notion of ϵ-variation between the two scoring functions. Another issue we plan to consider is the effect of lossless compression after our lossy compression, as compared to pure lossless compression.

Acknowledgments

We thank Oren Dinay for implementation of the algorithms described in this work.

6. REFERENCES

[1] C. Buckley and A. F. Lewit. Optimization of inverted vector searches. In *Proceedings of the Eighth International ACM-SIGIR Conference*, pages 97–110, Montreal, Canada, June 1985.

[2] C. Buckley, A. Singhal, M. Mitra, and G. Salton. New retrieval approaches using SMART: TREC 4. In *Proceedings of the Fourth Text REtrieval Conference (TREC-4)*, pages 25–48, Gaithersberg, Maryland, November 1995.

[3] S. Deerwester, S. Dumais, G. Furnas, T. Landauer, and R. Harshman. Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, 41(6):391–407, 1990.

[4] R. Fagin, R. Kumar, and D. Sivakumar. Top k orderings and near metrics. To appear, 2001.

[5] Pirate Search for Palm. `http://www.alphaworks.ibm.com/tech/piratesearch`.

[6] D. Harman and G. Candela. Retrieving records from a gigabyte of text on a minicomputer using statistical ranking. *Journal of the American Society of Information Science*, 41(8):581–589, 1990.

[7] M. Kendall and J. D. Gibbons. *Rank correlation methods*. Edward Arnold, London, 5th edition, 1990.

[8] Y. Maarek and F. Smadja. Full text indexing based on lexical relations: An application: Software libraries. In *Proceedings of the Twelfth International ACM SIGIR Conference*, pages 198–206, Cambridge, MA, June 1989.

[9] A. Moffat and J. Zobel. Fast ranking in limited space. In *Proceedings of the 10th IEEE International Conference on Data Engineering,*, pages 428–4376, Houston, TX, February 1994.

[10] M. Persin. Document filtering for fast ranking. In *Proceedings of the Seventeenth International ACM-SIGIR Conference*, pages 339–348, Dublin, Ireland, July 1994.

[11] M. Persin, J. Zobel, and R. Sacks-Davis. Filtered document retrieval with frequency-sorted indexes. *Journal of the American Society of Information Science*, 47(10):749–764, October 1996.

[12] G. Salton and M. J. McGill. *An Introduction to Modern Information Retrieval*. McGraw-Hill, New York, 1993.

[13] H. Turtle and J. Flood. Query evaluation: Strategies and optimizations. *Information Processing and Management*, 31(6):831–850, 1995.

[14] A. N. Vo and A. Moffat. Compressed inverted files with reduced decoding overheads. In *Proceedings of the 21st International ACM-SIGIR Conference*, pages 290–297, Melbourne, Australia, August 1998.

[15] E. M. Voorhees and D. K. Harman. Overview of the Seventh Text REtrieval Conference (TREC-7). In *Proceedings of the Seventh Text REtrieval Conference (TREC-7)*. National Institute of Standards and Technology, `http://trec.nist.gov/pubs.html`, 1999.

[16] I. H. Witten, A. Moffat, and T. C. Bell. *Managing Gigabytes*. Morgan Kaufman, San Francisco, 1999.