Pruning Policies for Two-Tiered Inverted Index with Correctness Guarantee

Alexandros Ntoulas[*]
Microsoft Search Labs
1065 La Avenida
Mountain View, CA 94043, USA
antoulas@microsoft.com

Junghoo Cho[†]
UCLA Computer Science Dept.
Boelter Hall
Los Angeles, CA 90095, USA
cho@cs.ucla.edu

ABSTRACT

The Web search engines maintain large-scale inverted indexes which are queried thousands of times per second by users eager for information. In order to cope with the vast amounts of query loads, search engines prune their index to keep documents that are likely to be returned as top results, and use this pruned index to compute the first batches of results. While this approach can improve performance by reducing the size of the index, if we compute the top results only from the pruned index we may notice a significant degradation in the result quality: if a document should be in the top results but was not included in the pruned index, it will be placed behind the results computed from the pruned index. Given the fierce competition in the online search market, this phenomenon is clearly undesirable.

In this paper, we study how we can avoid *any* degradation of result quality due to the pruning-based performance optimization, while still realizing most of its benefit. Our contribution is a number of modifications in the pruning techniques for creating the pruned index and a new result computation algorithm that *guarantees* that the top-matching pages are *always* placed at the top search results, even though we are computing the first batch from the pruned index most of the time. We also show how to determine the optimal size of a pruned index and we experimentally evaluate our algorithms on a collection of 130 million Web pages.

Categories and Subject Descriptors

H.3.1 [**Information Storage and Retrieval**]: Content Analysis and Indexing; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval

General Terms

Algorithms, Measuring, Performance, Design, Experimentation

Keywords

Inverted index, pruning, correctness guarantee, Web search engines

1. INTRODUCTION

The amount of information on the Web is growing at a prodigious rate [24]. According to a recent study [13], it is estimated that the Web currently consists of more than 11 billion pages. Due to this immense amount of available information, the users are becoming more and more dependent on the Web search engines for locating relevant information on the Web. Typically, the Web search engines, similar to other information retrieval applications, utilize a data structure called *inverted index*. An *inverted index* provides for the efficient retrieval of the documents (or Web pages) that contain a particular keyword.

In most cases, a query that the user issues may have thousands or even millions of matching documents. In order to avoid overwhelming the users with a huge amount of results, the search engines present the results in batches of 10 to 20 relevant documents. The user then looks through the first batch of results and, if she doesn't find the answer she is looking for, she may potentially request to view the next batch or decide to issue a new query.

A recent study [16] indicated that approximately 80% of the users examine at most the first 3 batches of the results. That is, 80% of the users typically view at most 30 to 60 results for every query that they issue to a search engine. At the same time, given the size of the Web, the inverted index that the search engines maintain can grow very large. Since the users are interested in a small number of results (and thus are viewing a small portion of the index for every query that they issue), using an index that is capable of returning all the results for a query may constitute a significant waste in terms of time, storage space and computational resources, which is bound to get worse as the Web grows larger over time [24].

One natural solution to this problem is to create a small index on a *subset* of the documents that are likely to be returned as the top results (by using, for example, the pruning techniques in [7, 20]) and compute the first batch of answers using the pruned index. While this approach has been shown to give significant improvement in performance, it also leads to noticeable degradation in the quality of the search results, because the top answers are computed only from the pruned index [7, 20]. That is, even if a page should be placed as the top-matching page according to a search engine's ranking metric, the page may be placed behind the ones contained in the pruned index if the page did not become part of the pruned index for various reasons [7, 20]. Given the fierce competition among search engines today this degradation is clearly undesirable and needs to be addressed if possible.

In this paper, we study how we can avoid *any* degradation of search quality due to the above performance optimization while still realizing most of its benefit. That is, we present a number of simple (yet important) changes in the pruning techniques for creating the pruned index. Our main contribution is a new answer computation algorithm that *guarantees* that the top-matching pages (according to the search-engine's ranking metric) are *always* placed at the top of search results, even though we are computing the first batch of answers from the pruned index most of the time. These enhanced pruning techniques and answer-computation algorithms are explored in the context of the *cluster architecture* commonly employed by today's search engines. Finally, we study and present how search engines can minimize the operational cost of answering queries while providing high quality search results.

[*]Work done while author was at UCLA Computer Science Department.

[†]This work is partially supported by NSF grants, IIS-0534784, IIS-0347993, and CNS-0626702. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the funding institutions.

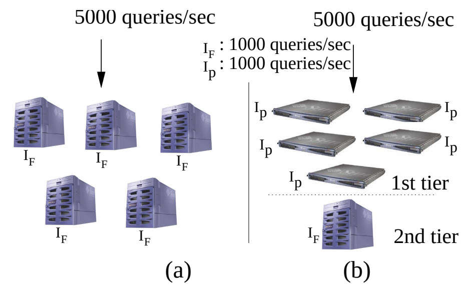

Figure 1: (a) Search engine replicates its full index I_F to increase query-answering capacity. (b) In the 1^{st} tier, small p-indexes I_P handle most of the queries. When I_P cannot answer a query, it is redirected to the 2^{nd} tier, where the full index I_F is used to compute the answer.

2. CLUSTER ARCHITECTURE AND COST SAVINGS FROM A PRUNED INDEX

Typically, a search engine downloads documents from the Web and maintains a local *inverted index* that is used to answer queries quickly.

Inverted indexes. Assume that we have collected a set of documents $\mathcal{D} = \{D_1, \ldots, D_M\}$ and that we have extracted all the terms $\mathcal{T} = \{t_1, \ldots, t_n\}$ from the documents. For every single term $t_i \in \mathcal{T}$ we maintain a list $I(t_i)$ of document IDs that contain t_i. Every entry in $I(t_i)$ is called a posting and can be extended to include additional information, such as how many times t_i appears in a document, the positions of t_i in the document, whether t_i is bold/italic, etc. The set of all the lists $I = \{I(t_1), \ldots, I(t_n)\}$ is our inverted index.

2.1 Two-tier index architecture

Search engines are accepting an enormous number of queries every day from eager users searching for relevant information. For example, Google is estimated to answer more than 250 million user queries per day. In order to cope with this huge query load, search engines typically replicate their index across a large cluster of machines as the following example illustrates:

Example 1 Consider a search engine that maintains a cluster of machines as in Figure 1(a). The size of its full inverted index I_F is larger than what can be stored in a single machine, so each copy of I_F is stored across four different machines. We also suppose that one copy of I_F can handle the query load of 1000 queries/sec. Assuming that the search engine gets 5000 queries/sec, it needs to replicate I_F five times to handle the load. Overall, the search engine needs to maintain $4 \times 5 = 20$ machines in its cluster. □

While fully replicating the entire index I_F multiple times is a straightforward way to scale to a large number of queries, typical query loads at search engines exhibit certain localities, allowing for significant reduction in cost by replicating only a small portion of the full index. In principle, this is typically done by pruning a full index I_F to create a smaller, *pruned index (or p-index)* I_P, which contains a subset of the documents that are likely to be returned as top results.

Given the p-index, search engines operate by employing a two-tier index architecture as we show in Figure 1(b): All incoming queries are first directed to one of the p-indexes kept in the 1^{st} tier. In the cases where a p-index cannot compute the answer (e.g. was unable to find enough documents to return to the user) the query is answered by redirecting it to the 2^{nd} tier, where we maintain a full index I_F. The following example illustrates the potential reduction in the query-processing cost by employing this two-tier index architecture.

Example 2 Assume the same parameter settings as in Example 1. That is, the search engine gets a query load of 5000 queries/sec and every copy of an index (both the full I_F and p-index I_P) can handle up to 1000 queries/sec. Also assume that the size of I_P is one fourth of I_F and thus can be stored on a single machine. Finally, suppose that the p-indexes can handle 80% of the user queries by themselves and only forward the remaining 20% queries to I_F. Under this setting, since all 5000/sec user queries are first directed to a p-index, five copies of I_P are needed in the 1^{st} tier. For the 2^{nd} tier, since 20% (or 1000 queries/sec) are forwarded, we need to maintain one copy of I_F to handle the load. Overall we need a total of 9 machines (five machines for the five copies of I_P and four machines for one copy of I_F). Compared to Example 1, this is more than 50% reduction in the number of machines. □

```
Algorithm 2.1   Computation of answer with correctness guarantee
  Input $q = (\{t_1, \ldots, t_n\}, [i, i+k])$ where
          $\{t_1, \ldots, t_n\}$: keywords in the query
          $[i, i+k]$: range of the answer to return
  Procedure
  (1)  $(\mathcal{A}, \mathcal{C})$ = ComputeAnswer$(q, I_P)$
  (2)  If $(\mathcal{C} = 1)$ Then
  (3)      Return $\mathcal{A}$
  (4)  Else
  (5)      $\mathcal{A}$ = ComputeAnswer$(q, I_F)$
  (6)      Return $\mathcal{A}$
```

Figure 2: Computing the answer under the two-tier architecture with the result correctness guarantee.

The above example demonstrates the potential cost saving achieved by using a p-index. However, the two-tier architecture may have a significant drawback in terms of its result quality compared to the full replication of I_F; given the fact that the p-index contains only a subset of the data of the full index, it is possible that, for some queries, the p-index may not contain the top-ranked document according to the particular ranking criteria used by the search engine and fail to return it as the top page, leading to noticeable quality degradation in search results. Given the fierce competition in the online search market, search engine operators desperately try to avoid any reduction in search quality in order to maximize user satisfaction.

2.2 Correctness guarantee under two-tier architecture

How can we avoid the potential degradation of search quality under the two-tier architecture? Our basic idea is straightforward: *We use the top-k result from the p-index only if we know for sure that the result is the same as the top-k result from the full index.* The algorithm in Figure 2 formalizes this idea. In the algorithm, when we compute the result from I_P (Step 1), we compute not only the top-k result $\mathcal{A}$, but also the *correctness indicator function* $\mathcal{C}$ defined as follows:

Definition 1 (Correctness indicator function) Given a query q, the p-index I_P returns the answer $\mathcal{A}$ together with a correctness indicator function $\mathcal{C}$. $\mathcal{C}$ is set to 1 if $\mathcal{A}$ is guaranteed to be *identical* (i.e. same results in the same order) to the result computed from the full index I_F. If it is possible that $\mathcal{A}$ is different, $\mathcal{C}$ is set to 0. □

Note that the algorithm returns the result from I_P (Step 3) only when it is identical to the result from I_F (condition $\mathcal{C} = 1$ in Step 2). Otherwise, the algorithm recomputes and returns the result from the full index I_F (Step 5). Therefore, the algorithm is guaranteed to return the same result as the full replication of I_F all the time.

Now, the real challenge is to find out (1) how we can compute the correctness indicator function $\mathcal{C}$ and (2) how we should prune the index to make sure that the majority of queries are handled by I_P alone.

Question 1 *How can we compute the correctness indicator function $\mathcal{C}$?*

A straightforward way to calculate $\mathcal{C}$ is to compute the top-k answer both from I_P and I_F and compare them. This naive solution, however, incurs a cost even higher than the full replication of I_F because the answers are computed *twice*: once from I_P and once from I_F. Is there any way to compute the correctness indicator function $\mathcal{C}$ only from I_P without computing the answer from I_F?

Question 2 *How should we prune I_F to I_P to realize the maximum cost saving?*

The effectiveness of Algorithm 2.1 critically depends on how often the correctness indicator function $\mathcal{C}$ is evaluated to be 1. If $\mathcal{C} = 0$ for all queries, for example, the answers to all queries will be computed twice, once from I_P (Step 1) and once from I_F (Step 5), so the performance will be *worse* than the full replication of I_F. What will be the optimal way to prune I_F to I_P, such that $\mathcal{C} = 1$ for a large fraction of queries? In the next few sections, we try to address these questions.

3. OPTIMAL SIZE OF THE P-INDEX

Intuitively, there exists a clear tradeoff between the size of I_P and the fraction of queries that I_P can handle: When I_P is large and has more information, it will be able to handle more queries, but the cost for maintaining and looking up I_P will be higher. When I_P is small, on the other hand, the cost for I_P will be smaller, but more queries will be forwarded to I_F, requiring us to maintain more copies of I_F. Given this tradeoff, how should we determine the optimal size of I_P in order to maximize the cost saving? To find the answer, we start with a simple example.

Example 3 Again, consider a scenario similar to Example 1, where the query load is 5000 queries/sec, each copy of an index can handle 1000 queries/sec, and the full index spans across 4 machines. But now, suppose that if we prune I_F by 75% to I_{P1} (i.e., the size of I_{P1} is 25% of I_F), I_{P1} can handle 40% of the queries (i.e., $\mathcal{C} = 1$ for 40% of the queries). Also suppose that if I_F is pruned by 50% to I_{P2}, I_{P2} can handle 80% of the queries. Which one of the I_{P1}, I_{P2} is preferable for the 1^{st}-tier index?

To find out the answer, we first compute the number of machines needed when we use I_{P1} for the 1^{st} tier. At the 1^{st} tier, we need 5 copies of I_{P1} to handle the query load of 5000 queries/sec. Since the size of I_{P1} is 25% of I_F (that requires 4 machines), one copy of I_{P1} requires one machine. Therefore, the total number of machines required for the 1^{st} tier is $5 \times 1 = 5$ (5 copies of I_{P1} with 1 machine per copy). Also, since I_{P1} can handle 40% of the queries, the 2^{nd} tier has to handle 3000 queries/sec (60% of the 5000 queries/sec), so we need a total of $3 \times 4 = 12$ machines for the 2^{nd} tier (3 copies of I_F with 4 machines per copy). Overall, when we use I_{P1} for the 1^{st} tier, we need $5 + 12 = 17$ machines to handle the load. We can do similar analysis when we use I_{P2} and see that a total of 14 machines are needed when I_{P2} is used. Given this result, we can conclude that using I_{P2} is preferable. □

The above example shows that the cost of the two-tier architecture depends on two important parameters: the size of the p-index and the fraction of the queries that can be handled by the 1^{st} tier index alone. We use s to denote the size of the p-index relative to I_F (i.e., if $s = 0.2$, for example, the p-index is 20% of the size of I_F). We use $f(s)$ to denote the fraction of the queries that a p-index of size s can handle (i.e., if $f(s) = 0.3$, 30% of the queries return the value $\mathcal{C} = 1$ from I_P). In general, we can expect that $f(s)$ will increase as s gets larger because I_P can handle more queries as its size grows. In Figure 3, we show an example graph of $f(s)$ over s.

Given the notation, we can state the problem of p-index-size optimization as follows. In formulating the problem, we assume that the number of machines required to operate a two-tier architecture

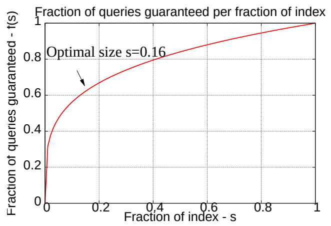

Figure 3: Example function showing the fraction of guaranteed queries $f(s)$ at a given size s of the p-index.

is roughly proportional to the total size of the indexes necessary to handle the query load.

Problem 1 (Optimal index size) Given a query load Q and the function $f(s)$, find the optimal p-index size s that minimizes the total size of the indexes necessary to handle the load Q. □

The following theorem shows how we can determine the optimal index size.

Theorem 1 *The cost for handling the query load Q is minimal when the size of the p-index, s, satisfies $\frac{d\,f(s)}{d\,s} = 1$.* □

Proof The proof of this and the following theorems is omitted due to space constraints.

This theorem shows that the optimal point is when the slope of the $f(s)$ curve is 1. For example, in Figure 3, the optimal size is when $s = 0.16$. Note that the exact shape of the $f(s)$ graph may vary depending on the query load and the pruning policy. For example, even for the same p-index, if the query load changes significantly, fewer (or more) queries may be handled by the p-index, decreasing (or increasing)$f(s)$. Similarly, if we use an effective pruning policy, more queries will be handled by I_P than when we use an ineffective pruning policy, increasing $f(s)$. Therefore, the function $f(s)$ and the optimal-index size may change significantly depending on the query load and the pruning policy. In our later experiments, however, we find that even though the shape of the $f(s)$ graph changes noticeably between experiments, the optimal index size consistently lies between 10%–30% in most experiments.

4. PRUNING POLICIES

In this section, we show how we should prune the full index I_F to I_P, so that (1) we can compute the correctness indicator function $\mathcal{C}$ from I_P itself and (2) we can handle a large fraction of queries by I_P. In designing the pruning policies, we note the following two localities in the users' search behavior:

1. **Keyword locality:** Although there are many different words in the document collection that the search engine indexes, a few popular keywords constitute the majority of the query loads. This *keyword locality* implies that the search engine will be able to answer a significant fraction of user queries even if it can handle only these few popular keywords.
2. **Document locality:** Even if a query has millions of matching documents, users typically look at only the first few results [16]. Thus, as long as search engines can compute the first few top-k answers correctly, users often will not notice that the search engine actually has not computed the correct answer for the remaining results (unless the users explicitly request them).

Based on the above two localities, we now investigate two different types of pruning policies: (1) a *keyword pruning* policy, which takes advantage of the keyword locality by pruning the whole inverted list $I(t_i)$ for unpopular keywords t_i's and (2) a *document pruning* policy, which takes advantage of the document locality by keeping only a few postings in each list $I(t_i)$, which are likely to be included in the top-k results.

As we discussed before, we need to be able to compute the correctness indicator function from the pruned index alone in order to provide the correctness guarantee. Since the computation of correctness indicator function may critically depend on the particular ranking function used by a search engine, we first clarify our assumptions on the ranking function.

4.1 Assumptions on ranking function

Consider a query $q = \{t_1, t_2, \ldots, t_w\}$ that contains a subset of the index terms. The goal of the search engine is to return the documents that are most relevant to query q. This is done in two steps: first we use the inverted index to find all the documents that contain the terms in the query. Second, once we have the relevant documents, we calculate the rank (or score) of each one of the documents with respect to the query and we return to the user the documents that rank the highest.

Most of the major search engines today return documents containing *all* query terms (i.e. they use AND-semantics). In order to make our discussions more concise, we will also assume the popular AND-semantics while answering a query. It is straightforward to extend our results to OR-semantics as well. The exact ranking function that search engines employ is a closely guarded secret. What is known, however, is that the factors in determining the document ranking can be roughly categorized into two classes:

Query-dependent relevance. This particular factor of relevance captures how relevant the query is to every document. At a high level, given a document D, for every term t_i a search engine assigns a *term relevance score* $tr(D, t_i)$ to D. Given the $tr(D, t_i)$ scores for every t_i, then the *query-dependent relevance* of D to the query, noted as $tr(D, q)$, can be computed by combining the individual term relevance values. One popular way for calculating the query–dependent relevance is to represent both the document D and the query q using the TF.IDF vector space model [29] and employ a *cosine distance* metric.

Since the exact form of $tr(D, t_i)$ and $tr(D, q)$ differs depending on the search engine, we will not restrict to any particular form; instead, in order to make our work applicable in the general case, we will make the generic assumption that the query-dependent relevance is *computed as a function* of the individual term relevance values in the query:

$$tr(D, q) = f_{tr}(tr(D, t_1), \ldots, tr(D, t_w)) \qquad (1)$$

Query-independent document quality. This is a factor that measures the overall "quality" of a document D *independent* of the particular query issued by the user. Popular techniques that compute the general quality of a page include PageRank [26], HITS [17] and the likelihood that the page is a "spam" page [25, 15]. Here, we will use $pr(D)$ to denote this query-independent part of the final ranking function for document D.

The final ranking score $r(D, q)$ of a document will depend on both the query-dependent and query-independent parts of the ranking function. The exact combination of these parts may be done in a variety of ways. In general, we can assume that the final ranking score of a document is a *function* of its query-dependent and query-independent relevance scores. More formally:

$$r(D, q) = f_r(tr(D, q), pr(D)) \qquad (2)$$

For example, $f_r(tr(D, q), pr(D))$ may take the form $f_r(tr(D, q), pr(D)) = \alpha \cdot tr(D, q) + (1 - \alpha) \cdot pr(D)$, thus giving weight α to the query-dependent part and the weight $1 - \alpha$ to the query-independent part.

In Equations 1 and 2 the exact form of f_r and f_{tr} can vary depending on the search engine. Therefore, to make our discussion applicable independent of the particular ranking function used by search engines, in this paper, we will make only the generic assumption that the ranking function $r(D, q)$ is monotonic on its parameters $tr(D, t_1), \ldots, tr(D, t_w)$ and $pr(D)$.

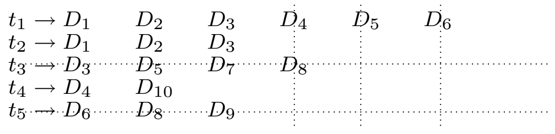
$t_1 \rightarrow D_1$ D_2 D_3 D_4 D_5 D_6
$t_2 \rightarrow D_1$ D_2 D_3
$t_3 \rightarrow D_3$ D_5 D_7 D_8
$t_4 \rightarrow D_4$ D_{10}
$t_5 \rightarrow D_6$ D_8 D_9

Figure 4: Keyword and document pruning.

```
Algorithm 4.1    Computation of $\mathcal{C}$ for keyword pruning
  Procedure
  (1)  $\mathcal{C} = 1$
  (2)  Foreach $t_i \in q$
  (3)      If $(I(t_i) \notin I_P)$ Then $\mathcal{C} = 0$
  (4)  Return $\mathcal{C}$
```

Figure 5: Result guarantee in keyword pruning.

Definition 2 A function $f(\alpha, \beta, \ldots, \omega)$ is monotonic if $\forall \alpha_1 \geq \alpha_2, \forall \beta_1 \geq \beta_2, \ldots \; \forall \omega_1 \geq \omega_2$ it holds that: $f(\alpha_1, \beta_1, \ldots, \omega_1) \geq f(\alpha_2, \beta_2, \ldots, \omega_2)$.

Roughly, the monotonicity of the ranking function implies that, between two documents D_1 and D_2, if D_1 has higher query-dependent relevance than D_2 and also a higher query-independent score than D_2, then D_1 should be ranked higher than D_2, which we believe is a reasonable assumption in most practical settings.

4.2 Keyword pruning

Given our assumptions on the ranking function, we now investigate the "keyword pruning" policy, which prunes the inverted index I_F "horizontally" by removing the whole $I(t_i)$'s corresponding to the least frequent terms. In Figure 4 we show a graphical representation of keyword pruning, where we remove the inverted lists for t_3 and t_5, assuming that they do not appear often in the query load.

Note that after keyword pruning, if all keywords $\{t_1, \ldots, t_n\}$ in the query q appear in I_P, the p-index has the same information as I_F as long as q is concerned. In other words, if all keywords in q appear in I_P, the answer computed from I_P is guaranteed to be the same as the answer computed from I_F. Figure 5 formalizes this observation and computes the correctness indicator function $\mathcal{C}$ for a keyword-pruned index I_P. It is straightforward to prove that the answer from I_P is identical to that from I_F if $\mathcal{C} = 1$ in the above algorithm.

We now consider the issue of optimizing the I_P such that it can handle the largest fraction of queries. This problem can be formally stated as follows:

Problem 2 (Optimal keyword pruning) Given the query load Q and a goal index size $s \cdot |I_F|$ for the pruned index, select the inverted lists $I_P = \{I(t_1), \ldots, I(t_h)\}$ such that $|I_P| \leq s \cdot |I_F|$ and the fraction of queries that I_P can answer (expressed by $f(s)$) is maximized. □

Unfortunately, the optimal solution to the above problem is intractable as we can show by reducing from knapsack (we omit the complete proof).

Theorem 2 *The problem of calculating the optimal keyword pruning is NP-hard.* □

Given the intractability of the optimal solution, we need to resort to an approximate solution. A common approach for similar knapsack problems is to adopt a greedy policy by keeping the items with the *maximum benefit per unit cost* [9]. In our context, the potential benefit of an inverted list $I(t_i)$ is the number of queries that can be answered by I_P when $I(t_i)$ is included in I_P. We approximate this number by the fraction of queries in the query load Q that include the term t_i and represent it as $P(t_i)$. For example, if 100 out of 1000 queries contain the term `computer`,

```
Algorithm 4.2    Greedy keyword pruning $HS$
  Procedure
  (1)  $\forall t_i$, calculate $HS(t_i) = \frac{P(t_i)}{|I(t_i)|}$.
  (2)  Include the inverted lists with the highest
       $HS(t_i)$ values such that $|I_P| \leq s \cdot |I_F|$.
```

Figure 6: Approximation algorithm for the optimal keyword pruning.

```
Algorithm 4.3    Global document pruning $VS_G$
  Procedure
  (1)  Sort all documents $D_i$ based on $pr(D_i)$
  (2)  Find the threshold value $\tau_p$, such that
       only $s$ fraction of the documents have $pr(D_i) > \tau_p$
  (4)  Keep $D_i$ in the inverted lists if $pr(D_i) > \tau_p$
```

Figure 7: Global document pruning based on pr.

then $P(computer) = 0.1$. The cost of including $I(t_i)$ in the p-index is its size $|I(t_i)|$. Thus, in our greedy approach in Figure 6, we include $I(t_i)$'s in the decreasing order of $P(t_i)/|I(t_i)|$ as long as $|I_P| \leq s \cdot |I_F|$. Later in our experiment section, we evaluate what fraction of queries can be handled by I_P when we employ this greedy keyword-pruning policy.

4.3 Document pruning

At a high level, *document pruning* tries to take advantage of the observation that most users are mainly interested in viewing the top few answers to a query. Given this, it is unnecessary to keep all postings in an inverted list $I(t_i)$, because users will not look at most of the documents in the list anyway. We depict the conceptual diagram of the document pruning policy in Figure 4. In the figure, we "vertically prune" postings corresponding to D_4, D_5 and D_6 of t_1 and D_8 of t_3, assuming that these documents are unlikely to be part of top-k answers to user queries. Again, our goal is to develop a pruning policy such that (1) we can compute the correctness indicator function $\mathcal{C}$ from I_P alone and (2) we can handle the largest fraction of queries with I_P. In the next few sections, we discuss a few alternative approaches for document pruning.

4.3.1 Global PR-based pruning

We first investigate the pruning policy that is commonly used by existing search engines. The basic idea for this pruning policy is that the query-independent quality score $pr(D)$ is a very important factor in computing the final ranking of the document (e.g. PageRank is known to be one of the most important factors determining the overall ranking in the search results), so we build the p-index by keeping only those documents whose pr values are high (i.e., $pr(D) > \tau_p$ for a threshold value τ_p). The hope is that most of the top-ranked results are likely to have high $pr(D)$ values, so the answer computed from this p-index is likely to be similar to the answer computed from the full index. Figure 7 describes this pruning policy more formally, where we sort all documents D_i's by their respective $pr(D_i)$ values and keep a D_i in the p-index when its $pr(D_i)$ value is higher than the *global threshold* value τ_p. We refer to this pruning policy as *global PR-based pruning (GPR)*.

```
Algorithm 4.4    Local document pruning $VS_L$
    N: maximum size of a single posting list
  Procedure
  (1)  Foreach $I(t_i) \in I_F$
  (2)      Sort $D_i$'s in $I(t_i)$ based on $pr(D_i)$
  (3)      If $|I(t_i)| \leq N$ Then keep all $D_i$'s
  (4)      Else keep the top-$N$ $D_i$'s with the highest $pr(D_i)$
```

Figure 8: Local document pruning based on pr.

```
Algorithm 4.5    Extended keyword-specific document pruning
  Procedure
  (1)  For each $I(t_i)$
  (2)      Keep $D \in I(t_i)$ if $pr(D) > \tau_{pi}$ or $tr(D, t_i) > \tau_{ti}$
```

Figure 9: Extended keyword-specific document pruning based on pr and tr.

Variations of this pruning policy are possible. For example, we may adjust the threshold value τ_p *locally* for each inverted list $I(t_i)$, so that we maintain *at least a certain number of postings for each inverted list $I(t_i)$*. This policy is shown in Figure 8. We refer to this pruning policy as *local PR-based pruning (LPR)*. Unfortunately, the biggest shortcoming of this policy is that we can prove that we cannot compute the correctness function $\mathcal{C}$ from I_P alone when I_P is constructed this way.

Theorem 3 *No PR-based document pruning can provide the result guarantee.* □

Proof Assume we create I_P based on the GPR policy (generalizing the proof to LPR is straightforward) and that every document D with $pr(D) > \tau_p$ is included in I_P. Assume that the k^{th} entry in the top-k results, has a ranking score of $r(D_k, q) = f_r(tr(D_k, q), pr(D_k))$. Now consider another document D_j that was pruned from I_P because $pr(D_j) < \tau_p$. Even so, it is still possible that the document's $tr(D_j, q)$ value is very high such that $r(D_j, q) = f_r(tr(D_j, q), pr(D_j)) > r(D_k, q)$. ■

Therefore, under a PR-based pruning policy, the quality of the answer computed from I_P can be significantly worse than that from I_F and it is not possible to detect this degradation without computing the answer from I_F. In the next section, we propose simple yet essential changes to this pruning policy that allows us to compute the correctness function $\mathcal{C}$ from I_P alone.

4.3.2 Extended keyword-specific pruning

The main problem of global PR-based document pruning policies is that we do not know the term-relevance score $tr(D, t_i)$ of the pruned documents, so a document not in I_P may have a higher ranking score than the ones returned from I_P because of their high tr scores.

Here, we propose a new pruning policy, called *extended keyword-specific document pruning (EKS)*, which avoids this problem by pruning not just based on the query-independent $pr(D)$ score but also based on the term-relevance $tr(D, t_i)$ score. That is, for every inverted list $I(t_i)$, we pick two threshold values, τ_{pi} for pr and τ_{ti} for tr, such that if a document $D \in I(t_i)$ satisfies $pr(D) > \tau_{pi}$ or $tr(D, t_i) > \tau_{ti}$, we include it in $I(t_i)$ of I_P. Otherwise, we prune it from I_P. Figure 9 formally describes this algorithm. The threshold values, τ_{pi} and τ_{ti}, may be selected in a number of different ways. For example, if pr and tr have equal weight in the final ranking and if we want to keep at most N postings in each inverted list $I(t_i)$, we may want to set the two threshold values equal to τ_i ($\tau_{pi} = \tau_{ti} = \tau_i$) and adjust τ_i such that N postings remain in $I(t_i)$.

This new pruning policy, when combined with a monotonic scoring function, enables us to compute the correctness indicator function $\mathcal{C}$ from the pruned index. We use the following example to explain how we may compute $\mathcal{C}$.

Example 4 Consider the query $q = \{t_1, t_2\}$ and a monotonic ranking function, $f(pr(D), tr(D, t_1), tr(D, t_2))$. There are three possible scenarios on how a document D appears in the pruned index I_P.

1. *D appears in both $I(t_1)$ and $I(t_2)$ of I_P*: Since complete information of D appears in I_P, we can compute the exact

```
Algorithm 4.6    Computing Answer from $I_P$
  Input Query $q = \{t_1, \ldots, t_w\}$
  Output $\mathcal{A}$: top-$k$ result, $\mathcal{C}$: correctness indicator function
  Procedure
  (1)  For each $D_i \in I(t_1) \cup \cdots \cup I(t_w)$
  (2)      For each $t_m \in q$
  (3)          If $D_i \in I(t_m)$
  (4)              $tr^*(D_i, t_m) = tr(D_i, t_m)$
  (5)          Else
  (6)              $tr^*(D_i, t_m) = \tau_{tm}$
  (7)          $f(D_i) = f(pr(D_i), tr^*(D_i, t_1), \ldots, tr^*(D_i, t_n))$
  (8)  $\mathcal{A}$ = top-$k$ $D_i$'s with highest $f(D_i)$ values
  (9)  $\mathcal{C} = \begin{cases} 1 & \text{if all } D_i \in \mathcal{A} \text{ appear in all } I(t_i), t_i \in q \\ 0 & \text{otherwise} \end{cases}$
```

Figure 10: Ranking based on thresholds $tr_\tau(t_i)$ and $pr_\tau(t_i)$.

score of D based on $pr(D)$, $tr(D, t_1)$ and $tr(D, t_2)$ values in I_P: $f(pr(D), tr(D, t_1), tr(D, t_2))$.

2. *D appears only in $I(t_1)$ but not in $I(t_2)$*: Since D does not appear in $I(t_2)$, we do not know $tr(D, t_2)$, so we cannot compute its exact ranking score. However, from our pruning criteria, we know that $tr(D, t_2)$ cannot be larger than the threshold value τ_{t2}. Therefore, from the monotonicity of f (Definition 2), we know that the ranking score of D, $f(pr(D), tr(D, t_1), tr(D, t_2))$, cannot be larger than $f(pr(D), tr(D, t_1), \tau_{t2})$.

3. *D does not appear in any list*: Since D does not appear at all in I_P, we do not know any of the $pr(D)$, $tr(D, t_1)$, $tr(D, t_2)$ values. However, from our pruning criteria, we know that $pr(D) \le \tau_{p1}$ and $\le \tau_{p2}$ and that $tr(D, t_1) \le \tau_{t1}$ and $tr(D, t_2) \le \tau_{t2}$. Therefore, from the monotonicity of f, we know that the ranking score of D, cannot be larger than $f(min(\tau_{p1}, \tau_{p2}), \tau_{t1}, \tau_{t2})$. □

The above example shows that when a document does not appear in one of the inverted lists $I(t_i)$ with $t_i \in q$, we cannot compute its exact ranking score, but we can still compute its *upper bound* score by using the threshold value τ_{ti} for the missing values. This suggests the algorithm in Figure 10 that computes the top-k result $\mathcal{A}$ from I_P together with the correctness indicator function $\mathcal{C}$. In the algorithm, the correctness indicator function $\mathcal{C}$ is set to one only if all documents in the top-k result $\mathcal{A}$ appear in all inverted lists $I(t_i)$ with $t_i \in q$, so we know their exact score. In this case, because these documents have scores higher than the upper bound scores of any other documents, we know that no other documents can appear in the top-k. The following theorem formally proves the correctness of the algorithm. In [11] Fagin et al., provides a similar proof in the context of multimedia middleware.

Theorem 4 *Given an inverted index I_P pruned by the algorithm in Figure 9, a query $q = \{t_1, \ldots, t_w\}$ and a monotonic ranking function, the top-k result from I_P computed by Algorithm 4.6 is the same as the top-k result from I_F if $\mathcal{C} = 1$.* □

Proof Let us assume D_k is the k^{th} ranked document computed from I_P according to Algorithm 4.6. For every document $D_i \in I_F$ that is not in the top-k result from I_P, there are two possible scenarios:

First, D_i is not in the final answer because it was pruned from all inverted lists $I(t_j), 1 \le j \le w$, in I_P. In this case, we know that $pr(D_i) \le min_{1 \le j \le w} \tau_{pj} < pr(D_k)$ and that $tr(D_i, t_j) \le \tau_{tj} < tr(D_k, t_j), 1 \le j \le w$. From the monotonicity assumption, it follows that the ranking score of D_I is $r(D_i) < r(D_k)$. That is, D_i's score can never be larger than that of D_k.

Second, D_i is not in the answer because D_i is pruned from some inverted lists, say, $I(t_1), \ldots, I(t_m)$, in I_P. Let us assume $\bar{r}(D_i) = f(pr(D_i), \tau_{t1}, \ldots, \tau_{tm}, tr(D_i, t_{m+1}), \ldots, tr(D_i, t_w))$. Then, from $tr(D_i, t_j) \le \tau_{tj} (1 \le j \le m)$ and the monotonicity assumption,

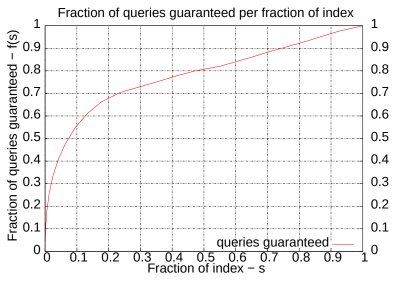

Figure 11: Fraction of *guaranteed queries* $f(s)$ answered in a keyword-pruned p-index of size s.

we know that $r(D_i) \le \bar{r}(D_i)$. Also, Algorithm 4.6 sets $\mathcal{C} = 1$ only when the top-k documents have scores larger than $\bar{r}(D_i)$. Therefore, $r(D_i)$ cannot be larger than $r(D_k)$. ■

5. EXPERIMENTAL EVALUATION

In order to perform realistic tests for our pruning policies, we implemented a search engine prototype. For the experiments in this paper, our search engine indexed about 130 million pages, crawled from the Web during March of 2004. The crawl started from the Open Directory's [10] homepage and proceeded in a breadth-first manner. Overall, the total uncompressed size of our crawled Web pages is approximately 1.9 TB, yielding a full inverted index I_F of approximately 1.2 TB.

For the experiments reported in this section we used a real set of queries issued to Looksmart [22] on a daily basis during April of 2003. After keeping only the queries containing keywords that were present in our inverted index, we were left with a set of about 462 million queries. Within our query set, the average number of terms per query is 2 and 98% of the queries contain at most 5 terms.

Some experiments require us to use a particular ranking function. For these, we use the ranking function similar to the one used in [20]. More precisely, our ranking function $r(D, q)$ is

$$r(D, q) = pr_{norm}(D) + tr_{norm}(D, q) \qquad (3)$$

where $pr_{norm}(D)$ is the normalized PageRank of D computed from the downloaded pages and $tr_{norm}(D, q)$ is the normalized TF.IDF cosine distance of D to q. This function is clearly simpler than the real functions employed by commercial search engines, but we believe for our evaluation this simple function is adequate, because we are not studying the effectiveness of a ranking function, but the effectiveness of pruning policies.

5.1 Keyword pruning

In our first experiment we study the performance of the keyword pruning, described in Section 4.2. More specifically, we apply the algorithm HS of Figure 6 to our full index I_F and create a *keyword-pruned* p-index I_P of size s. For the construction of our keyword-pruned p-index we used the query frequencies observed during the first 10 days of our data set. Then, using the remaining 20-day query load, we measured $f(s)$, the fraction of queries handled by I_P. According to the algorithm of Figure 5, a query can be handled by I_P (i.e., $\mathcal{C} = 1$) if I_P includes the inverted lists for *all* of the query's keywords.

We have repeated the experiment for varying values of s, picking the keywords greedily as discussed in Section 4.2.The result is shown in Figure 11. The horizontal axis denotes the size s of the p-index as a fraction of the size of I_F. The vertical axis shows the fraction $f(s)$ of the queries that the p-index of size s can answer. The results of Figure 11, are very encouraging: we can answer a significant fraction of the queries with a small fraction of the original index. For example, approximately 73% of the queries can be answered using 30% of the original index. Also, we find that when we use the keyword pruning policy only, the optimal index size is $s = 0.17$.

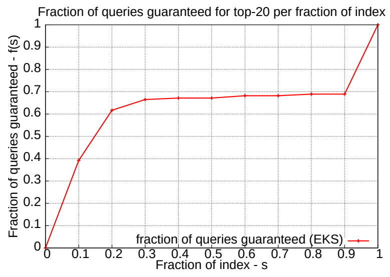

Figure 12: Fraction of *guaranteed* queries $f(s)$ answered in a document-pruned p-index of size s.

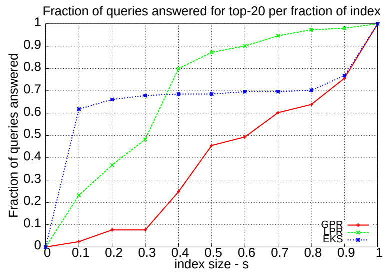

Figure 13: Fraction of queries answered in a document-pruned p-index of size s.

5.2 Document pruning

We continue our experimental evaluation by studying the performance of the various document pruning policies described in Section 4.3. For the experiments on document pruning reported here we worked with a 5.5% sample of the whole query set. The reason behind this is merely practical: since we have much less machines compared to a commercial search engine it would take us about a year of computation to process all 462 million queries.

For our first experiment, we generate a *document-pruned* p-index of size s by using the Extended Keyword-Specific pruning (EKS) in Section 4. Within the p-index we measure the fraction of queries that can be guaranteed (according to Theorem 4) to be correct. We have performed the experiment for varying index sizes s and the result is shown in Figure 12. Based on this figure, we can see that our document pruning algorithm performs well across the scale of index sizes s: for all index sizes larger than 40%, we can guarantee the correct answer for about 70% of the queries. This implies that our EKS algorithm can successfully identify the necessary postings for calculating the top-20 results for 70% of the queries by using at least 40% of the full index size. From the figure, we can see that the optimal index size $s = 0.20$ when we use EKS as our pruning policy.

We can compare the two pruning schemes, namely the keyword pruning and EKS, by contrasting Figures 11 and 12. Our observation is that, if we would have to pick one of the two pruning policies, then the two policies seem to be more or less equivalent for the p-index sizes $s \leq 20\%$. For the p-index sizes $s > 20\%$, keyword pruning does a much better job as it provides a higher number of guarantees at any given index size. Later in Section 5.3, we discuss the combination of the two policies.

In our next experiment, we are interested in comparing EKS with the PR-based pruning policies described in Section 4.3. To this end, apart from EKS, we also generated *document-pruned* p-indexes for the *Global pr-based pruning (GPR)* and the *Local pr-based pruning (LPR)* policies. For each of the polices we created document-pruned p-indexes of varying sizes s. Since GPR and LPR cannot provide a correctness guarantee, we will compare the fraction of queries from each policy that are *identical* (i.e. the same results in the same order) to the top-k results calculated from the full index. Here, we will report our results for $k = 20$; the results are similar for other values of k. The results are shown in Figure 13.

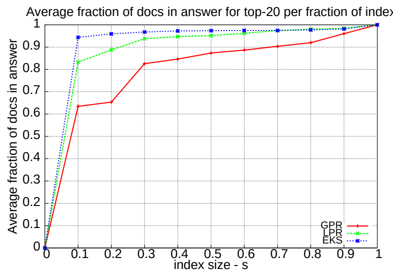

Figure 14: Average fraction of the top-20 results of p-index with size s contained in top-20 results of the full index.

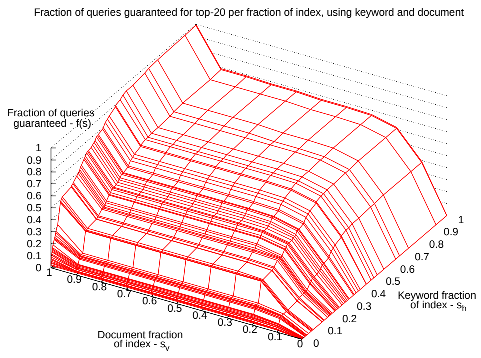

Figure 15: Combining keyword and document pruning.

The horizontal axis shows the size s of the p-index; the vertical axis shows the fraction $f(s)$ of the queries whose top-20 results are identical to the top-20 results of the full index, for a given size s.

By observing Figure 13, we can see that GPR performs the worst of the three policies. On the other hand EKS, picks up early, by answering a great fraction of queries (about 62%) correctly with only 10% of the index size. The fraction of queries that LPR can answer remains below that of EKS until about $s = 37\%$. For any index size larger than 37%, LPR performs the best.

In the experiment of Figure 13, we applied the strict definition that the results of the p-index have to be in the same order as the ones of the full index. However, in a practical scenario, it may be acceptable to have some of the results out of order. Therefore, in our next experiment we will measure the fraction of the results coming from an p-index that are contained within the results of the full index. The result of the experiment is shown on Figure 14. The horizontal axis is, again, the size s of the p-index; the vertical axis shows the average fraction of the top-20 results common with the top-20 results from the full index. Overall, Figure 14 depicts that EKS and LPR identify the same high ($\approx 96\%$) fraction of results on average for any size $s \geq 30\%$, with GPR not too far behind.

5.3 Combining keyword and document pruning

In Sections 5.1 and 5.2 we studied the individual performance of our keyword and document pruning schemes. One interesting question however is how do these policies perform in combination? What fraction of queries can we guarantee if we apply both keyword and document pruning in our full index I_F?

To answer this question, we performed the following experiment. We started with the full index I_F and we applied keyword pruning to create an index I_P^h of size $s_h \cdot 100\%$ of I_F. After that, we further applied document pruning to I_P^h, and created our final p-index I_P of size $s_v \cdot 100\%$ of I_P^h. We then calculated the fraction of guaranteed queries in I_P. We repeated the experiment for different values of s_h and s_v. The result is shown on Figure 15. The x-axis shows the index size s_h after applying keyword pruning; the y-axis shows the index size s_v after applying document pruning; the z-axis shows the fraction of guaranteed queries after the two prunings. For example the point (0.2, 0.3, 0.4) means that if we apply keyword pruning and keep 20% of I_F, and subsequently on the resulting index we apply document pruning keeping 30% (thus creating a p-index of size $20\% \cdot 30\% = 6\%$ of I_F) we can guarantee 40% of the queries. By observing Figure 15, we can see that for p-index sizes smaller than 50%, our combined pruning does relatively well. For example, by performing 40% keyword and 40% document pruning (which translates to a pruned index with $s = 0.16$) we can provide a guarantee for about 60% of the queries. In Figure 15, we also observe a "plateau" for $s_h > 0.5$ and $s_v > 0.5$. For this combined pruning policy, the optimal index size is at $s = 0.13$, with $s_h = 0.46$ and $s_v = 0.29$.

6. RELATED WORK

[3, 30] provide a good overview of *inverted indexing* in Web search engines and IR systems. Experimental studies and analyses of various *partitioning schemes* for an inverted index are presented in [6, 23, 33]. The pruning algorithms that we have presented in this paper are independent of the partitioning scheme used.

The works in [1, 5, 7, 20, 27] are the most related to ours, as they describe pruning techniques based on the idea of keeping the postings that contribute the most in the final ranking. However, [1, 5, 7, 27] do not consider any query-independent quality (such as PageRank) in the ranking function. [32] presents a generic framework for computing approximate top-k answers with some probabilistic bounds on the quality of results. Our work essentially extends [1, 2, 4, 7, 20, 27, 31] by proposing mechanisms for providing the correctness guarantee to the computed top-k results.

Search engines use various methods of *caching* as a means of reducing the cost associated with queries [18, 19, 21, 31]. This thread of work is also orthogonal to ours because a caching scheme may operate on top of our p-index in order to minimize the answer computation cost. The exact *ranking functions* employed by current search engines are closely guarded secrets. In general, however, the rankings are based on query-dependent relevance and query-independent document "quality." Query-dependent relevance can be calculated in a variety of ways (see [3, 30]). Similarly, there are a number of works that measure the "quality" of the documents, typically as captured through link-based analysis [17, 28, 26]. Since our work does not assume a particular form of ranking function, it is complementary to this body of work.

There has been a great body of work on *top-k result calculation*. The main idea is to either stop the traversal of the inverted lists early, or to shrink the lists by pruning postings from the lists [14, 4, 11, 8]. Our proof for the correctness indicator function was primarily inspired by [12].

7. CONCLUDING REMARKS

Web search engines typically prune their large-scale inverted indexes in order to scale to enormous query loads. While this approach may improve performance, by computing the top results from a pruned index we may notice a significant degradation in the result quality. In this paper, we provided a framework for new pruning techniques and answer computation algorithms that *guarantee* that the top matching pages are *always* placed at the top of search results in the correct order. We studied two pruning techniques, namely keyword-based and document-based pruning as well as their combination. Our experimental results demonstrated that our algorithms can effectively be used to prune an inverted index without degradation in the quality of results. In particular, a keyword-pruned index can guarantee 73% of the queries with a size of 30% of the full index, while a document-pruned index can guarantee 68% of the queries with the same size. When we combine the two pruning algorithms we can guarantee 60% of the queries with an index size of 16%. It is our hope that our work will help search engines develop better, faster and more efficient indexes and thus provide for a better user search experience on the Web.

8. REFERENCES

[1] V. N. Anh, O. de Kretser, and A. Moffat. Vector-space ranking with effective early termination. In *SIGIR*, 2001.

[2] V. N. Anh and A. Moffat. Pruning strategies for mixed-mode querying. In *CIKM*, 2006.

[3] R. A. Baeza-Yates and B. A. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press / Addison-Wesley, 1999.

[4] N. Bruno, L. Gravano, and A. Marian. Evaluating top-k queries over web-accessible databases. In *ICDE*, 2002.

[5] S. Büttcher and C. L. A. Clarke. A document-centric approach to static index pruning in text retrieval systems. In *CIKM*, 2006.

[6] B. Cahoon, K. S. McKinley, and Z. Lu. Evaluating the performance of distributed architectures for information retrieval using a variety of workloads. *ACM TOIS*, 18(1), 2000.

[7] D. Carmel, D. Cohen, R. Fagin, E. Farchi, M. Herscovici, Y. Maarek, and A. Soffer. Static index pruning for information retrieval systems. In *SIGIR*, 2001.

[8] S. Chaudhuri and L. Gravano. Optimizing queries over multimedia repositories. In *SIGMOD*, 1996.

[9] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms, 2nd Edition*. MIT Press/McGraw Hill, 2001.

[10] Open directory. http://www.dmoz.org.

[11] R. Fagin. Combining fuzzy information: an overview. In *SIGMOD Record, 31(2)*, 2002.

[12] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *PODS*, 2001.

[13] A. Gulli and A. Signorini. The indexable web is more than 11.5 billion pages. In *WWW*, 2005.

[14] U. Guntzer, G. Balke, and W. Kiessling. Towards efficient multi-feature queries in heterogeneous environments. In *ITCC*, 2001.

[15] Z. Gyöngyi, H. Garcia-Molina, and J. Pedersen. Combating web spam with trustrank. In *VLDB*, 2004.

[16] B. J. Jansen and A. Spink. An analysis of web documents retrieved and viewed. In *International Conf. on Internet Computing*, 2003.

[17] J. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, September 1999.

[18] R. Lempel and S. Moran. Predictive caching and prefetching of query results in search engines. In *WWW*, 2003.

[19] R. Lempel and S. Moran. Optimizing result prefetching in web search engines with segmented indices. *ACM Trans. Inter. Tech.*, 4(1), 2004.

[20] X. Long and T. Suel. Optimized query execution in large search engines with global page ordering. In *VLDB*, 2003.

[21] X. Long and T. Suel. Three-level caching for efficient query processing in large web search engines. In *WWW*, 2005.

[22] Looksmart inc. `http://www.looksmart.com`.

[23] S. Melnik, S. Raghavan, B. Yang, and H. Garcia-Molina. Building a distributed full-text index for the web. *ACM TOIS*, 19(3):217–241, 2001.

[24] A. Ntoulas, J. Cho, C. Olston. What's new on the web? The evolution of the web from a search engine perspective. In *WWW*, 2004.

[25] A. Ntoulas, M. Najork, M. Manasse, and D. Fetterly. Detecting spam web pages through content analysis. In *WWW*, 2006.

[26] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford University.

[27] 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), 1996.

[28] M. Richardson and P. Domingos. The intelligent surfer: Probabilistic combination of link and content information in pagerank. In *Advances in Neural Information Processing Systems*, 2002.

[29] S. Robertson and K. Spärck-Jones. Relevance weighting of search terms. *Journal of the American Society for Information Science*, 27:129–46, 1976.

[30] G. Salton and M. J. McGill. *Introduction to modern information retrieval*. McGraw-Hill, first edition, 1983.

[31] P. C. Saraiva, E. S. de Moura, N. Ziviani, W. Meira, R. Fonseca, and B. Riberio-Neto. Rank-preserving two-level caching for scalable search engines. In *SIGIR*, 2001.

[32] M. Theobald, G. Weikum, and R. Schenkel. Top-k query evaluation with probabilistic guarantees. In *VLDB*, 2004.

[33] A. Tomasic and H. Garcia-Molina. Performance of inverted indices in shared-nothing distributed text document information retrieval systems. In *Parallel and Distributed Information Systems*, 1993.