Improving Web Search Efficiency via a Locality Based Static Pruning Method

Edleno S. de Moura
Federal University of Amazonas, Brazil
edleno@dcc.ufam.edu.br

Célia F. dos Santos
Federal University of Amazonas, Brazil
cfs@dcc.ufam.edu.br

Daniel R. Fernandes
Federal University of Amazonas, Brazil
drf@dcc.ufam.edu.br

Altigran S. Silva
Federal University of Amazonas, Brazil
alti@dcc.ufam.edu.br

Pavel Calado
INESC-ID, Portugal
pavel@algos.inesc-id.pt

Mario A. Nascimento
University of Alberta, Canada
mn@cs.ualberta.ca

ABSTRACT

The unarguably fast, and continuous, growth of the volume of indexed (and indexable) documents on the Web poses a great challenge for search engines. This is true regarding not only search effectiveness but also time and space efficiency. In this paper we present an index pruning technique targeted for search engines that addresses the latter issue without disconsidering the former. To this effect, we adopt a new pruning strategy capable of greatly reducing the size of search engine indices. Experiments using a real search engine show that our technique can reduce the indices' storage costs by up to 60% over traditional lossless compression methods, while keeping the loss in retrieval precision to a minimum. When compared to the indices size with no compression at all, the compression rate is higher than 88%, i.e., less than one eighth of the original size. More importantly, our results indicate that, due to the reduction in storage overhead, query processing time can be reduced to nearly 65% of the original time, with no loss in average precision. The new method yields significative improvements when compared against the best known static pruning method for search engine indices. In addition, since our technique is orthogonal to the underlying search algorithms, it can be adopted by virtually any search engine.

Categories and Subject Descriptors

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

General Terms

Performance, Experimentation

Keywords

pruning, indexing, search engines, web search, information retrieval

WWW 2005, May 10-14, 2005, Chiba, Japan.
ACM 1-59593-046-9/05/0005.

1. INTRODUCTION

Search engines are an essential resource for making the Web useful, and usable, for the public in general. In fact, recent reports indicate that Web traffic due to search engines has doubled from 2002 to 2003[1]. The volume of Web pages available keeps also growing at a fast pace, fostered mainly by the increasing number of people interested in, and capable of, using the Web. Thus, there is a great demand for solutions that allow search engines to improve their performance, both in terms of effectiveness as well as efficiency. In this paper we aim at the latter without losing sight of the former. We propose and investigate a technique that can be used by *any* search engine to improve its efficiency in terms of query processing time and resource consumption, at nearly no cost in terms of the quality of query results.

To allow efficient query processing over the large amount of documents stored in its document database, search engines rely on a set of data structures –the *indices* [20]. At least two types of indices are usually deployed: The *frequency index*, which contains, for each term t, its frequency on each document where t occurs, and the *positional index*, which contains, for each term t, information about the occurrence positions of t on each document. The first allows us to measure the relative importance of terms in both the indexed documents as in the queries. The second allows search systems to process positional queries, such as phrase or proximity queries, and is usually the most expensive one to maintain and use. As almost all of the computational cost for processing user queries is due to the access to these indices, there is a great need for solutions that not only decrease storage costs, but also speed up the overall system performance by attenuating disk access. This must be achieved while retaining the quality of the answers provided by the search engine to its users.

In this paper, we present a new lossy compression method that is specifically designed for web search engines, taking into account typical features present in such systems. When compared to other lossy compression methods, our method achieves better ranking quality levels with a similar compression rate. In other words, it is possible to reach bet-

[1] http://www.clickz.com/news/article.php/2108921.

ter compression rates while maintaining the quality of the search results.

To support our claims, we present experiments which show that our method allows a 60% reduction in index sizes, where the indices were already compressed using traditional lossless compression methods. This represents not only a drastic reduction in storage costs but also a reduction of nearly 40% in average time to process each query, with almost no loss in the quality of ranking results.

To evaluate potential losses in raking quality, we have also performed experiments to compare the rankings produced by a search engine under the following circumstances: (1) using only a lossless compression method; (2) using the best known lossy compression method in the literature; and (3) using our method. We have found that the rankings produced by our method are more similar to the rankings generated without compression. At a compression rate of 76%, our method achieved results similar to the best known lossy compression method with a compression rate of 60%. For phrase queries, the gain was even higher, at an 85% compression rate the results were similar to those obtained by the best known lossy compression method at a 57% compression rate.

This paper is structured as follows. Section 2 discusses the motivation for compressing search engine indices and presents work related to this topic. In particular, we discuss the baseline method used in our experiments, which was extended here to deal with practical situations not considered previously. Section 3 presents details of our proposal. Section 4 presents our experimental setup and the results obtained with a real web search engine. Section 5 discusses important practical issues not explicitly mentioned in the experiments, concerning the deployment of pruning-based lossy compression methods on web search engines. Finally, Section 6 concludes the paper and presents suggestions for future work.

2. COMPRESSING SEARCH INDICES

On typical search engines, the frequency and the positional indices require about the same amount of storage space required by the document database. For instance, TodoBR[2], one of the largest Brazilian search engines, has a document database that requires 22 Gb to store approximately 10 million documents, while its indices require a total of roughly 31.7 Gb when stored with no compression, of which 26.3 Gb are for the positional index and 5.4 Gb for the frequency index. In addition, almost all of the computational cost associated with processing user queries is due to the time needed to access these indices. Thus, a reduction in their size not only decreases storage costs but also speeds up the overall system performance. In essence, if the search engine indices are smaller, there will be less data to be read from disk and thus less time spent on query processing.

A successful approach to reduce the index size is the use of data compression techniques. These techniques can be lossless [1, 20, 3, 7, 16], where no information is ever discarded and the original files can be obtained from their compressed version, or lossy, where the index size is reduced by discarding information that is deemed not useful at query processing time. The use of lossless compression, however, imposes upper bounds on the compression rates, which represents a practical limit to the performance improvements and reduction in storage space that can achieved.

To overcome such bounds, one can use lossy compression methods. Examples of such methods are stop-word removal, where the index entries for terms that are very common are removed, and Latent Semantic Indexing (LSI) [8]. Although these methods have been proposed originally to remove noisy information from the indices, they can be seen as lossy compression techniques, since they also reduce indices sizes. These two methods, however, have strong drawbacks. LSI involves high computational costs, rendering its application not practical for search engines that deal with non-trivial size collections. Stop-word removal typically does help in improving search performance, but it is a quite limited compression technique since it only removes inverted lists for a restricted set of terms and is often domain dependent. An *inverted list* is the complete set of entries for a term in the index. This technique usually yields just a small reduction in index sizes.

A more effective lossy compression alternative for search indices is the use of *pruning methods* [1, 17, 5]. The rationale is to remove from the indices entries whose potential contribution to the relevance score of the documents in which they occur is so small that their removal will have little effect in the final ranking position of these documents. The expected impact of this removal is a noticeable reduction in storage requirements, I/O operations and computational overhead. Our proposal fits in this category. Next we present a brief review of pruning methods for search engine indices, focusing on a particular technique [5], which we used as a departure point for our proposal.

2.1 Pruning Methods

Pruning methods have been used to reduce both CPU costs and the amount of elements read from disk during query processing. Such methods can be classified as *dynamic* [17, 1] and *static* [5]. Dynamic methods maintain the index completely stored on disk and use heuristics to avoid reading unnecessary information at query processing time. In this case, the amount of pruning performed varies according to the user queries, which represents an advantage, since they can be better adapted to each specific query. In contrast, static methods try to predict, at index construction time, the entries which will not be useful at query processing time. These entries are then removed from the index and, for this reason, static methods can be seen as lossy compression methods. Static methods offer the advantage of both reducing the disk storage costs and time to process each query. A system that uses both static and dynamic methods can also be implemented to take advantage of the two types of pruning options.

We are interested in static pruning methods as a lossy compression alternative for search engine indices. Static pruning methods proposed and experimented in the literature so far take into account only disjunctive queries. It is this restriction that allows those methods to predict the individual impact of each single term in the final ranking score. This restriction, however, is not applicable to most web search engines, where conjunctive queries and phrases are usually allowed and are popular among users [2, 6]. These types of queries may have a significant impact on the performance of the pruning methods. For instance, to achieve a significant reduction in the time to process a phrase query,

[2]http://www.todobr.com.br

it is necessary to remove entries not only from the frequency index, but also from the positional index. However, all methods proposed so far only consider removing entries from the frequency index.

This lack of studies on how to use pruning strategies on real case search engines has motivated us, to not only study the performance of a previously proposed static pruning method in this scenario, but also to propose new pruning solutions specifically designed to deal with typical search engine queries.

2.2 Carmel's Pruning Method

A particularly successful static pruning method was proposed by Carmel et al. [5]. For simplicity, hereafter we refer to this method as *Carmel's* method. This method has demonstrated excellent practical results and, for this reason, we also chose it to serve as a baseline of comparison to our approach.

The list of all entries of a term t in frequency (and also in the positional) index is known as the *inverted list* of t. The main idea behind Carmel's method is to use the search engine's ranking to compute the importance of each inverted list and determine which entries of the index can be removed. This information is obtained by taking the individual terms from the collection vocabulary and submitting each of them as a single-term query to the search system. The resulting document list for each term t will then contain the documents which are related to t, sorted in decreasing order of importance according to the search engine's ranking criteria. Notice that this ranking criteria already includes any type of evidence (e.g., link information, anchor text, or others) that the search engine considers when ranking documents.

Let R_t be the resulting list for a term t. R_t provides a ranking for the indexed documents. The higher a document D ranks in R_t, the higher the chance that D will appear in the results of a query containing term t. Carmel's method takes just the top portion of R_t to guide the pruning process. Each frequency index entry that represents the occurrence of a term t in a document D is removed from the index if D is not present in the top portion of R_t.

The number of relevant entries in the inverted list of a term t in the frequency index is determined according to a criteria called δ-*top*, which in turn is based on a parameter $\delta \in [0, 1]$. The value of δ is typically chosen via experiments. The criteria δ-*top* only preserves entries that correspond to documents in R_t whose score, given by the search engine, is at least δ times the highest score of all documents in R_t. This new list is called $R_t(\delta)$. For instance, if $\delta = 0.7$, each document with a score of at least 70% of the top score will be in $R_t(\delta)$. Ideally, this criteria will preserve a number of elements large enough to give a good approximation of the ranking scores for the top answers of every query submitted to the system.

The original proposal of Carmel et al. does not indicate how the method could be applied to prune entries from positional indices. In our study, we extended the method to perform experiments with phrase queries. For this, we prune the positional index by removing all the occurrences that correspond to each entry pruned from the frequency index. For instance, if an entry for document D with frequency 4 is removed from the inverted list of term t, then the four entries that represent the occurrences of t in D are also removed from the positional index.

In Section 4, we present experimental results of Carmel's pruning method when applied to positional indices in phrase queries. Our experiments indicate that this method is in fact useful for many of the queries commonly submitted to web search engines, but that gains are not as significant as in the case of scenarios with a majority of conjunctive queries or phrase queries. Based on the observation of the weaknesses of this method, we propose and experiment a new alternative, designed specifically for pruning indices in a web search engine environment, considering not only disjunctive but also conjunctive and phrase queries.

3. LBPM: A NEW PRUNING METHOD

Some types of queries typically submitted to search engines, such as conjunctive queries and phrases, require that the pruning method preserves index entries for documents that occur in the inverted lists of different terms, i.e., documents where two query terms occur together. Since Carmel's method takes information *individually* from each term to prune the entries, the resulting pruned index may not hold this property. As a consequence, important documents may not be present at the final ranking.

This problem is illustrated in Figure 1, where the ranked list for term A, R_A, starts with documents 3, 15, 1, 8 and 14, and finishes with documents 2, 31, 4 and 13. The ranked list for term B, R_B, starts with documents 2, 1, 7, 9 and 43 and finishes with documents 25, 16, 3 and 21. Suppose that the shadowed areas correspond to portions pruned in both lists. After the pruning, a query requiring documents that contain both terms A *and* B would include document 1 in the answer, but would not include documents 2 and 3. However, these last two documents would be originally in the answer before pruning. Further, documents 2 and 3 could be important to the query, since they appear on the top positions of the individual rankings for terms A and B, respectively. This example shows that a pruning solution which does not take into account co-occurrences of terms within documents may fail to preserve important results for conjunctive and phrase queries.

Based on this observation, we propose a variation of Carmel's method that aims at predicting what set of terms may occur together in queries and uses this information to preserve common documents in the inverted lists of these terms. Therefore, besides top entries of each postings for each term, our method preserves also entries that are in the top list of other related terms.

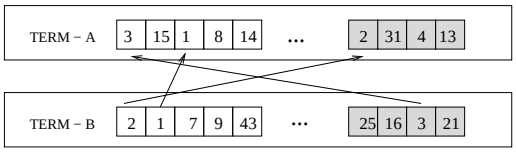

Figure 1: Loss of information caused by pruning term inverted lists without considering co-occurrence of terms across documents.

A key step for any pruning method is to predict the index entries that will be effectively useful at query processing time. In cases where queries are composed of conjunctive expressions or phrases, this means we also need to predict which sets of words will appear together in the queries.

An intuitive heuristic is to consider words that appear close in the documents as good candidates to appear together in a query. We have adopted this idea to produce a new pruning method, which we call the *Locality-based pruning method (lbpm)*. In our method, we consider that two words are close if they appear in the same sentence. Sentences are obtained from the documents by extracting fragments of text that represent natural language sentences.

As in Carmel's proposal, our method determines which term occurrences are individually important to each document. The method uses this information to select from the text database sentences that are associated with such occurrences, here called *significant sentences*. Following, the significant sentences are sorted and selected in order to control the final index sizes. Finally, they are used as the input information that guides the pruning process. In summary, our method comprises three steps: (1) determining significant sentences, (2) ranking the significant sentences and (3) pruning the indices. These steps are detailed in the following sections.

3.1 Determining Significant Sentences

Our goal in this step is to obtain sentences that, according to the search engine ranking criteria, are related to the most significant terms of each document. We use the method proposed by Carmel et al [5] as the starting point to obtain these sentences.

We start by determining the *significant terms* of each document D by computing, for each term t in the database vocabulary $\mathcal{V}$, the ranked list $R_t(\delta)$. Notice that this corresponds exactly to Carmel's pruning method. We can now define the significant terms of a document D as:

$$T(D) = \{t | D \in R_t(\delta), \forall t \in \mathcal{V}\}$$

Thus, the significant terms of a document D, denoted by $T(D)$ are all terms t whose set $R_t(\delta)$ contains D.

We use this information to compute the significant sentences of a document D, which are all those that contain at least one of the significant terms of D, i.e.:

$$S(D) = \{S_i | T(D) \cap S_i \neq \emptyset, \forall S_i \in D\}$$

where each S_i is a set of words corresponding to a sentence in D.

This alone could be used to filter out entries and produce the pruned index. However, in many applications it is important to keep control of the final size of the pruned index. In this case, since the index size is proportional to the number of significant sentences selected, it is necessary to select a limited number of sentences in order to keep the final index sizes under control. In order to add this form of control to our method, we rank the significant sentences of a document to determine a relevance level for each one of them. This ranking is then used to control the number of relevant sentences and, consequently the estimated final index sizes.

3.2 Selecting Sentences

Suppose that we want to select only a fixed number of significant sentences of D from set $S(D)$. This is achieved by ranking the sentences in $S(D)$ according to an estimation of their importance for search engine queries.

Consider the existence of a function $common(S(D), T(D))$ that returns the sentence in $S(D)$ that has the most terms in common with $T(D)$. The procedure presented in Figure 2 is used in our method to rank the sentences and select the ones which will guide the pruning process.

```
1  Sentence Selection
2  begin
3     let D be a document;
4     let T(D) be the set of significant terms for D;
5     let S(D) be the set of significant sentences for D;
6     let p be the final desired percentage;
8     T'(D) ← T(D);
9     S'(D) ← ∅;
10    size ← 0;
11    do
12        S_common ← common(S(D), T'(D));
13        S'(D) ← S'(D) ∪ S_common;
14        size ← size + |S_common|;
15        T'(D) ← T'(D) − {x|x ∈ S_common ∧ x ∈ T'(D)};
16        S(D) ← S(D) − S_common;
17        if T'(D) = ∅ then T'(D) ← T(D)
18    until (size ≥ p|D|) or S(D) = ∅
19    return S'(D);
20 end
```

Figure 2: Procedure for selecting a subset of significant sentences.

This procedure works by selecting sentences until there are no more significant sentences to select (i.e., $S(D)$ is empty) or until the size, in terms, of all the selected sentences reaches a percentage p of the size of D. The algorithm makes a copy of $T(D)$ to $T'(D)$, and then executes the loop in the lines from 9 to 16. Each iteration of the loop adds to $S'(D)$ the sentence S from $S(D)$ that has more terms in common with $T'(D)$. The *size* variable is then incremented with the size of S, the terms in S are removed from $T'(D)$ and S is removed from S(D). The removal of terms from $T'(D)$ allows selecting each sentence of $S'(D)$ based on different sets of significant terms. If the sentences already selected cover all significant terms, $T'(D)$ becomes empty, which makes the algorithm assign $T'(D) = T(D)$ (line 15) again, starting a new round of choices based on the significant terms, as long as there is room for adding new sentences given the threshold p.

The selection of sentences has the goal of preserving as many significant terms as possible. The extra cost given by the algorithm to perform this task is not high. The algorithm can be implemented in $O(s^2)$, where s is the number of sentences of each document. However, since the size of each document is independent of the number of indexed documents, s is constant per document and the cost to run the sentence selection algorithm for all documents is linear in the collection size.

3.3 Pruning the Indices

After selecting sentences to represent each document, these sentences guide the pruning process. In the positional index, only occurrences of terms in the selected sentences are preserved. The remaining are removed. In the frequency index, entries that represent the frequency of a term t in a document D are preserved only if t occurs at least once in at least one of the selected sentences of D.

4. EXPERIMENTS

In this section we provide experiments to evaluate the performance of the studied pruning methods. All experiments were carried out on a common search engine and show the relation between the amount of reduction in the index sizes obtained by each method, the eventual losses in the ranking quality, the divergence from the original search engine ranking, and the gain in time efficiency.

4.1 Experimental Setup

The experiments with pruning methods presented here were carried out on two collections. The Los Angeles Times (LAT) collection from TREC [13] and the TodoBR collection, which is a set of documents extracted from a real search engine. The LAT collection contains about 132,000 documents (467 MB). For this collection, the queries were selected from the ad-hoc tasks for TREC 8, for topics from 401 to 450. We have used the titles of the topics to compose short queries in order to have query sizes close to the ones typically used on the web. These titles were then applied to the search system as both disjunctive and conjunctive queries. Queries for LAT collection were processed using the traditional vector space model to rank the document answers [18, 1]. We have decided to use the LAT in the intent of discovering possible differences between running pruning methods on a web collection and on a non-web collection.

The TodoBR collection contains over 11 million web pages, collected from the Brazilian web. It was chosen to present the experiments in a real case search engine environment, in order to better validate the ideas presented here. We have used in the experiments queries extracted from a log of more than 1 million real user queries submitted to TodoBR. This log is one of the main reasons we have chosen TodoBR collection for our experiments, since it provides useful information about search engine user preferences.

All the indices used, including the original, i.e., not pruned indices, were compressed using *Elias-δ* lossless methods for coding document numbers, frequencies and positions, as described in [20]. The TodoBR index sizes after compression with *Elias-δ* are 8.4 Gb for the positional index and 1.4 Gb for the frequency index, totalizing 9.8 Gb of compressed indices. The compression rates computed for the lossy compression consider these index sizes as the baseline. For instance, a compression rate of 80% in the experiments with TodoBR means a total index size of 1.96 Gb. We have decided to combine lossless and lossy compression to show a scenario with fully compressed indices. The TodoBR indices with no compression at all require 31.7 Gb of disk storage, when representing the frequency entries with 8 bits, the document numbers with 28 bits and the occurrence positions with 28 bits. In the case of LAT collection, the indices with no compression require 738 MB of disk storage and the index sizes after compression with *Elias-δ* are 184.5 MB.

An important information about the experiments concerns the ranking strategies applied. As in every large scale search engine, the ranking algorithm applied on TodoBR was computed using not only the document texts, but also other auxiliary sources of evidence. For this ranking algorithm we have combined three different sources of information: document contents, anchor text concatenation and the authority value of each document.

The document content evidence, consists of the similarity between the text of the document and the user query, computed using the vector-space model [18, 1]. The anchor text concatenation evidence consists of the similarity between the anchor text in all linking documents and the user query, also computed using the vector-space model. The authority value evidence of each document is given by the *global* HITS algorithm. The *global* HITS algorithm consists of applying the HITS algorithm [14] to the full link graph of our web collection, instead of just to the documents related to the user query, as defined in [4]. The final similarity score of each document is given by:

$$s(d,q) = [1-(1-s_c(d,q))\times(1-s_a(d,q))\times(1-s_h(d,q))] \quad (1)$$

where s_c is the vector-space similarity of the query q with the contents of document d, s_a is the similarity of q with the anchor text concatenation associated with d, and s_h is the authority value of d.

Notice that the search engine ranking function is not our main focus here. We have decided to adopt a known solution proposed for search engines in order to have more realistic results in the experiments. As search engines usually take other information to compute the ranking, it would not be fair to perform experiments with only one source of evidence. Another important detail about the ranking used is that the main cost in both disk storage and computational effort comes from the indices we are pruning. Anchor text in the TodoBR collection has its own index that represents only 10% of the size of the main positional and frequency indices. The HITS values are pre-computed and fit in memory. More detailed information about this ranking combination can be obtained in [4].

Also, note that we are not interested in evaluating the quality of the ranking algorithm itself, but only the impact of the pruning method in the final ranking. We have also experimented with the system without this extra information, and the conclusions were the same as those obtained for the experiments shown here.

Finally, the experimental environment for evaluating the time efficiency of the systems comprises two machines running the Linux operating system version 2.4.21. The search engine server runs on a Pentium 1.7 GHz machine with 2 Gb of main memory, and three 36 Gb SCSI disks. The search engine client runs on a Pentium 4, 1 GHz machine with 512 Mb of main memory. The two machines are connected directly (using a crossover cable) by an 100-megabit fast Ethernet connection.

4.2 Query Types

Search engines usually provide rich query options to their users. For instance, Google[3] allows users to include query options like inserting phrases in the query or making a query term mandatory. In general, at least three types of queries are important for search engine environments: conjunctive, disjunctive, and phrase.

In search engines like Google, Altavista[4] and also in TodoBR, queries are taken as conjunctive by default and, as such, conjunctive queries are very common. In fact, in TodoBR, roughly 79% of the queries with more than one term are conjunctive. Phrase and disjunctive queries are much less common. For instance, in TodoBR they form 20% and 1% of the submitted queries, respectively. Nonetheless, we also

[3] `http://www.google.com`
[4] `http://www.av.com`

will investigate these types of query due to its possible importance to other search systems.

To understand the impact of the proposed pruning method over each query type, we have experimented with all the three query types described above. Each of the query sets used in the experiments have 1000 queries selected from the TodoBR log. All the queries were selected randomly. Queries with only one term were removed from these experiments, because both methods yield very similar results at the compression rates experimented. For sake of completeness, we have also experimented the systems with a query set of 1000 queries randomly selected from the TodoBR log, without any restriction of type or number of terms. The query set in this case was composed of 89.3% conjunctive queries, 10.2% phrases and 0.5% disjunctive queries, while single-term queries correspond to 35.9%.

In the case of the LAT TREC collection, we have expressed the query topics as conjunctive and disjunctive queries, taking their title as the query. It was not possible to use the queries as phrases in these experiments since most of them resulted in empty or insignificant result sets.

4.3 Performance Evaluation

In order to study the quality of pruning methods we use here two distinct measures: the distance between the original ranking and the ranking obtained by the pruned indices and precision and recall curves. The ranking distance is computed here using a variation of Kendall's *tau* method, proposed in [9] to compare the top k answers of two different rankings.

Kendall's *tau* method yields a score that lies between 0, when the two rankings are identical and $k(3k-1)/2$, when the top k answers provided are completely disjoint. To normalize the results we have used the formula $x' = 1 - \frac{2x}{k(3k-1)}$, where x is the non-normalized Kendall's *tau* result, as also done in [5]. Therefore, the final comparison of rankings presented here will vary from 1, representing equal rankings, to 0, representing completely different rankings. The higher the value of x', the more similar the rankings compared.

For the experiments, we have compared the top 20 answers in all graphics presented. We have chosen this number since search engine users usually view only the top 20 results. In fact, most of them require only the first 10 results. In TodoBR for 70% of the queries only the first 10 results are viewed [19] and for roughly 90% of the queries the top 20 results are viewed.

Since we deal with phrase and conjunctive queries, it is common to have queries that provide small result lists. In some cases the number of results is smaller than 20. For this reason, we need to adapt Kendall's *tau* measure, since it was originally proposed to compare lists with an equal number of answers. This is not a problem in studies comparing the effect of pruning over disjunctive queries, since it is possible to assure the two lists will have at least k answers in each, for a previously determined k [5]. However, since we are dealing also with conjunctive and phrase queries, the number of answers when using pruned indices can be smaller than the ones in the original answer. To deal with this case, when getting a smaller number of answers in the pruned indices, we add fake answers to this list. These fake entries consist of invalid and distinct document numbers.

This modification takes into consideration the fact that the smaller list has lost information that was present in the original answer. For instance, with this modification, it is possible to compare an empty answer, obtained with the pruned indices, with an original answer list with 20 elements. We simply add 20 fake answers to the empty list and get two completely different rankings, yielding Kendall's *tau* measure equal to 0.

The second measure used to evaluate the pruning methods is the precision/recall curves obtained by each query. Precision is the percentage of documents retrieved that are relevant to a given query. Recall is the percentage of relevant documents that were retrieved. To plot the precision/recall curves we computed a precision value for each of the eleven standard recall points: 0, 10, 20, ..., and 100%. We have determined the set of relevant documents for each query using the pooling method employed for the Web-based collection of TREC [11, 12]. For constructing the pools we have evaluated the first top 20 answers of each ranking for each query.

4.4 Results

This section presents the experimental results obtained with *lbpm* and *Carmel's* method. The section is divided in three parts: first we study the similarity between rankings before and after pruning, next we study the impact of pruning on precision and recall and, finally, we measure time efficiency.

4.4.1 Ranking Similarity

The following graphics show the reduction in similarity as the compression rate increases. Five compression rates were tested. Although we have tried to produce compression rates similar for each method, compression rates can not be completely controlled neither by *lbpm* nor *Carmel's* method. However, the five rates used are close enough to allow a fair comparison.

Figure 3 presents results for the TodoBR collection, on a set of disjunctive queries. In this case, *Carmel's* method produced the results more similar to the original system at all levels of compression. A little loss in the similarity was expected for *lbpm*, since it replaces entries from the top of the lists of each individual term to include entries that help in conjunctive and phrase queries. However, results obtained are very close to the ones obtained with *Carmel's* method. Further, at all rates of compression, the similarity of *lbpm* was superior to 0.93, which indicates that the ranking produced is very similar to the original. It is also important to remember that disjunctive queries are the less common query type in TodoBR and are probably not very popular in other typical web search engines either.

Figure 4 shows the results for conjunctive queries, which represent the most popular query type in TodoBR. In this case, *lbpm* yielded better results with all the compression levels tested. The results show that, using *lbpm*, it is possible to obtain a compression rate of roughly 76%, obtaining the same similarity that would be obtained by *Carmel's* method with a compression rate of roughly 60%. At a 50% compression rate both methods yield ranking results very similar to the original, showing that even this high compression level would be safe in terms of preserving the original ranking quality for conjunctive queries.

The differences between similarities obtained for disjunctive and conjunctive (Figures 3 and 4) queries show the im-

portance of studying pruning methods dealing with different query types.

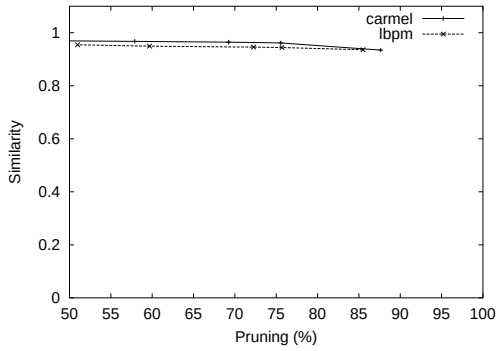

Figure 3: Kendall's *tau* ranking similarity obtained for *Carmel's* method and *lbpm* in the TodoBR collection, using only disjunctive queries.

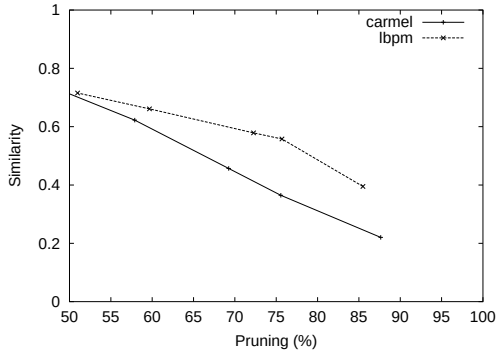

Figure 4: Kendall's *tau* ranking similarity obtained for *Carmel's* method and *lbpm* for the TodoBR collection, running only conjunctive queries and comparing the top 20 results.

The highest difference between the two methods was obtained for phrase queries. This result was expected, since *lbpm* performs a locality based pruning, meaning that it tends to preserve term occurrences if the terms appear close in the text. Figure 5 shows the results for phrase queries. *lbpm* yielded consistently better results in this case. For *Carmel's* method, results with a compression rate higher than 50% present very low similarities. *lbpm*, on the other hand, obtained a similarity of nearly 0.5 at an 85% compression rate. The highest compression rate that allowed such similarity for *Carmel's* method was 57%. At a 50% compression rate, *lbpm* yielded a similarity superior to 0.8 in phrases, while *Carmel's* method yielded less than 0.6%.

Figure 6 shows results obtained when processing all query types. Again *lbpm* yielded better results at all compression levels experimented. Notice that this graph shows results close to the ones obtained with conjunctive queries, since conjunctive queries are the most popular query type. The values for of *lbpm* and *Carmel's* method are closer in this graph since the two methods yielded similar results when processing single-term queries, which represent 35.9% of this query set.

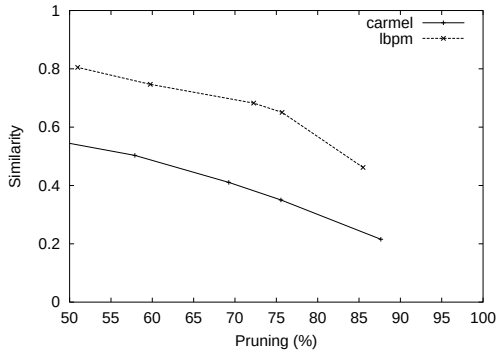

Figure 5: Kendall's *tau* ranking similarity obtained for *Carmel's* method and *lbpm* in the TodoBR collection, using only phrase queries and comparing the top 20 results.

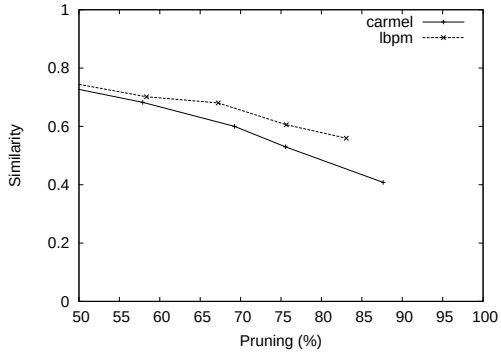

Figure 6: Kendall's *tau* ranking similarity obtained for *Carmel's* method and *lbpm* in TodoBR collection, using a set of 1000 random queries. The top 20 results were compared.

Figure 7 presents results for the LAT TREC collection, using 50 short queries extracted from topic titles and submitted as disjunctive and conjunctive queries. Results for LAT TREC are slightly different from the ones obtained for TodoBR. In this case, *Carmel's* method obtained better results for disjunctive queries with a significant difference. For conjunctive queries the two methods yielded almost the same results.

A closer observation of each query result shows that particular collection characteristics may have worsen results for *lbpm*. *Carmel's* method tends to preserve entries for terms that have shorter inverted lists. Many terms given in LAT TREC queries hold this property, contrary to terms in the TodoBR query log. This is probably due to the nature of the two collections. In LAT TREC, a rare term is usually correct and important to discriminate a document from others. For instance, the term 'osteoporosis' is present in the LAT TREC queries and appears only 44 times. Rare terms like this are common on the LAT TREC queries and are important for the collection.

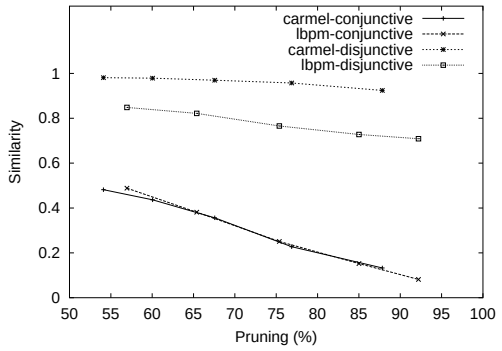

Figure 7: Kendall's *tau* ranking similarity obtained for *Carmel's* method and *lbpm* in LAT TREC collection, using disjunctive and conjunctive queries. The top 20 results were compared.

On the Web this is mostly not true, and most of the rare terms are usually nonsense, due to errors. Further, the collective nature of the Web makes terms that appear in queries have a higher chance of appearing in documents. This happens because a term that is important to a user is probably also important to other users. In other words, terms common in search engine queries tend also to be common on the web documents.

As a consequence, for the LAT TREC collection, *Carmel's* method preserves a significative portion of entries for terms that have appeared in the queries. On the other hand, *lbpm* tends to substitute entries of rare terms by entries of terms that appear together in selected sentences. In web collections this is not a great disadvantage, but in LAT TREC it is. We do not discard the hypothesis that other factors are interfering in the results and further study is left as future work. At any rate, these results emphasize the necessity of pruning methods specifically designed for web search engines, and also indicate that *lbpm* can be a good alternative in such cases.

4.4.2 Precision and Recall

In order to evaluate the impact of changes in the ranking when using the studied methods, we have performed experiments with the TodoBR collection submitting 40 queries extracted from the TodoBR logs. We randomly selected 32 conjunctive and 8 phrase non-single term queries from this log, and results were evaluated by a group of 15 people. The average number of answers in each query pool was 33 and the average number of relevant answers was 17.3.

Figure 8 shows the evolution of average precision as the compression rate increases, when using pruned indices generated by *lbpm* and *Carmel's* method. Average precision is computed by taking the average results of the eleven precision points for each ranking. At all compression levels *lbpm* values are closer to the ones obtained by the original indices (i.e., at 0% pruning). At 50% and 60% compression rate *lbpm* gives practically no loss in precision, while after that the losses become increasingly more apparent.

Figures 9 to 11 present the precision recall curves obtained with the original indices(*non-pruned*), and indices pruned with *lbpm* and *Carmel's* method at compression rates varying from 60% to 75%. At 60% compression rate, Figure 9, *lbpm* achieved results similar to those using non-pruned indices.

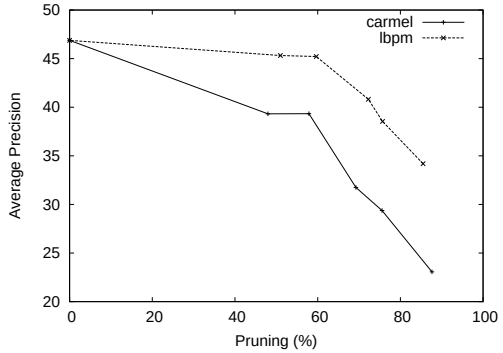

Figure 8: Average precision obtained when processing queries with indices pruned by *lbmp* and *Carmel's* method at different compression rates.

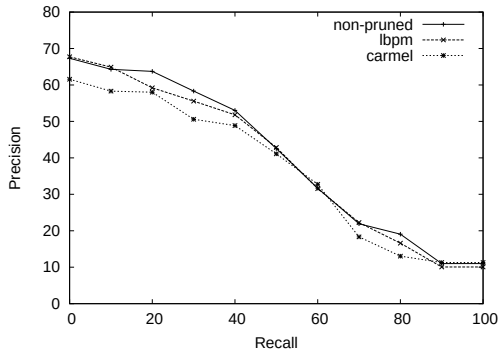

Figure 9: Precision recall curves for the TodoBR collection, using the original indices (*non-pruned*), and indices pruned by *lbpm* and *Carmel's* method with nearly 60% of compression rate.

Figure 10 shows that at 70% both *Carmel's* method and *lbpm* present losses in precision. However, *lbpm* achieved results considerably close to *non-pruned* for recall levels up to 50%. The top recall levels are the most important for search engines. Thus, in spite of the loss in average precision by 6 points, results indicate the losses at 70% compression rate for *lbpm* are acceptable. Finally, Figure 11 shows that both methods are no longer as effective in maintaining precision levels when the compression rates is higher than 75%.

These results allow us to conclude that, although the pruning process causes changes in the query rankings, as shown by Kendall's *tau* similarity, changes in precision are small. In practice, this implies that we can significantly reduce search engine index storage costs without compromising the users' satisfaction with the results.

4.4.3 Time Performance

Finally, we present experiments to evaluate time performance gains obtained with the reduction in the index sizes. Figure 12 presents the results obtained when processing queries at different compression rate levels using *lbpm* in

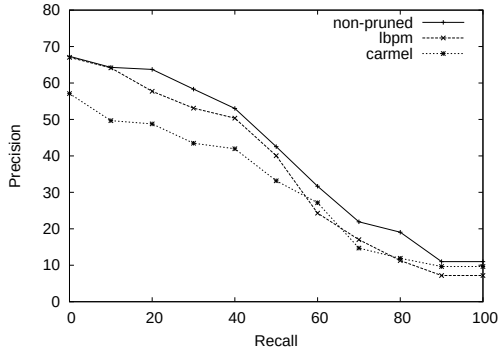

Figure 10: Precision recall curves for the TodoBR collection, using the original indices (*non-pruned*), and indices pruned by *lbpm* and *Carmel's* method with nearly 70% of compression rate.

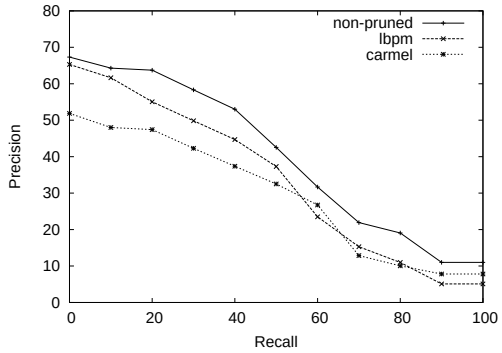

Figure 11: Precision recall curves for TodoBR collection, using the original indices (*non-pruned*), and indices pruned by *lbpm* and *Carmel's* method with nearly 75% of compression rate.

TodoBR. Time is given as a percentage of the time for processing the queries using the original non-pruned indices. Notice that the pruning method has reduced only the size of positional and frequency indices constructed for the text in the documents' body. Other index components of the search engine, such as the indices for the anchor text concatenation, were not pruned.

Obviously, as the pruned indices are reduced, the time for processing the queries is also reduced. This reduction follows an almost linear behavior up to a 70% compression rate. This is the expected gain curve if the index is considerably larger than the main memory available. At some point between 70% and 76%, the indices have become so small that the cache system causes a drop in the query processing time. For instance, at a 76% compression rate, the time for processing queries became roughly 20% of the time for processing the queries with the original index.

5. OTHER PRACTICAL ISSUES

One practical problem introduced with static pruning methods is the extra time the pruning process adds to index construction. For instance, the extra time to prune the indices

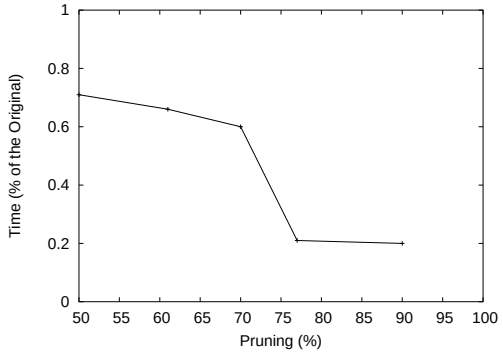

Figure 12: Time for processing a total of 100,000 queries extracted from the TodoBR query log, using different levels of compression. Time is given as a percentage of the time obtained for processing queries with the non-pruned indices.

with *Carmel's* method in these experiments was roughly half the time to construct the original indices. The *lbpm* algorithm used roughly the same time spent to construct the original indices. This relation is constant, since the time complexity of all these processes is linear in the collection size.

The extra time to prune indices could be prohibitive in some practical cases, since search engines continually update their collections and indices [15]. However, each update in a search engine collection changes only a small portion of the collection, since most web pages either are not visited by the search engine or their content did not change in the updating period. For instance, in a period of a week, it is expected that a only a small percentage of the collected web pages present any change in their text [10]. In this scenario, a simple solution to perform the update in the indices would be not pruning entries for the changed documents, adding all new indices entries that represent their new content without applying any pruning. This strategy might reduce the compression gain obtained by the pruning, since the compression rate would decrease slowly at each update.

Meanwhile, the search engine could update the pruning information, recomputing the pruned lists for the new collection at a larger time period, achieving again full compression. We believe the reduction in the computational costs to process queries can compensate the extra effort to keep this pruning update strategy in most practical cases. However, this topic needs a more detailed study and will certainly depend on the search engine architecture.

6. CONCLUSIONS AND FUTURE WORK

We presented a new lossy compression method for reducing search engine indices sizes with little impact on the quality of the answers to user queries. Our method uses a locality based heuristic for guiding the pruning process, which is capable of achieving high similarities with the original search engine rankings, even at high compression rates. Differently from previous work, which have considered only disjunctive queries, we have carried out experiments considering query types typically submitted to search engines: conjunctive and phrase queries.

With the proposed method, we were able to achieve a compression rate of 60%, while preserving a high similarity with the ranking computed using the original indices for all the three type of queries experimented. These results indicate pruning can be useful in practice for reducing the query processing and disk storage costs on search engines.

For future work, we intend to further study our method's efficiency in terms of time to generate the pruned indices. The time to generate the pruned indices in our current implementation is roughly the same time needed to create them. Although this is a reasonable amount of time, our index pruning implementation was not optimized and significant reduction can be achieved.

We have already obtained results capable of justifying the use of our approach on real web search engines. In spite of that, an interesting direction to be followed would be the study of incremental pruning techniques to reduce the cost of updating the search engine database and indices. Finally, we intend to study other techniques to predict occurrences of words together in a query. One of such approaches we are considering is the use of information derived from the search engine query log to improve the sentence selection process.

7. ACKNOWLEDGMENTS

This work is partially supported by CYTED VII.19 RIBIDI Project, GERINDO Project–grant MCT/CNPq/CT-INFO 552.087/02-5, CNPq individual grant 303576/2004-9 (Edleno S. de Moura), CAPES individual grant (Célia F. dos Santos), NSERC Canada (Mario Nascimento), SiteFix CNPq project and FAPEAM/PAPPE. We also thank Akwan Information Technologies for making TodoBR logs available for experiments.

8. REFERENCES

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

[2] D. Bahle, H. E. Williams, and J. Zobel. Efficient phrase querying with and auxiliary index. In *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 215–221, 2002.

[3] T. C. Bell, J. G. Cleary, and I. H. Witten. *Text compression*. Prentice Hall, 1990.

[4] P. P. Calado, E. S. de Moura, B. Ribeiro-Neto, I. Silva, and N. Ziviani. Local versus global link information in the web. *ACM Transactions on Information Systems (TOIS)*, 2003.

[5] D. Carmel, D. Cohen, R. Fagin, E. Farchi, M. Herscovici, Y. S. Maarek, and A. Soffer. Static index pruning for information retrieval systems. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 43–50, New Orleans, Louisiana, USA, September 2001.

[6] W. B. Croft, H. R. Turtle, and D. D. Lewis. The use of phrases and structured queries in information retrieval. In *Proceedings of the 14th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 32–45, 1991.

[7] E. S. de Moura, G. Navarro, N. Ziviani, and R. Baeza-Yates. Fast and flexible word searching on compressed text. *ACM Transactions on Information Systems(ACM TOIS)*, 18(2):113–139, April 2000.

[8] S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman. Indexing by latent semantic analysis. *Journal of American Society for Information Science*, 41(6), 1990.

[9] R. Fagin, R. Kumar, and D. Sivakumar. Comparing top k lists. In *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 28–36, 2003.

[10] D. Fetterly, M. Manasse, M. Najork, and J. Wiener. A large-scale study of the evolution of web page. In *Proceedings of the 12th International World Wide Web Conference*, pages 669–678, May 2003.

[11] D. Hawking, N. Craswell, and P. B. Thistlewaite. Overview of TREC-7 very large collection track. In *The Seventh Text REtrieval Conference (TREC-7)*, pages 91–104, Gaithersburg, Maryland, USA, November 1998.

[12] D. Hawking, N. Craswell, P. B. Thistlewaite, and D. Harman. Results and challenges in web search evaluation. *Computer Networks*, 31(11–16):1321–1330, May 1999. Also in Proceedings of the 8th International World Wide Web Conference.

[13] D. Hawking, E. Voorhees, P. Bailey, and N. Craswell. Overview of trec-8 web track. In *Proc. of TREC-8*, pages 131–150, Gaithersburg MD, November 1999.

[14] J. M. Kleinberg. Authoritative sources in a hyperlinked environment. In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 668–677, San Francisco, California, USA, January 1998.

[15] L. Lim, M. Wang, S. Padmanabhan, J. S. Vitter, and R. Agarwal. Dynamic maintenance of web indexes using landmarks. In *Proceedings of the 12th International World Wide Web Conference*, pages 102–111, May 2003.

[16] G. Navarro, E. S. de Moura, M. Neubert, N. Ziviani, and R. Baeza-Yates. Fast and flexible word searching on compressed text. *Information Retrieval Journal*, 3(1):49–77, 2000.

[17] 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, Oct. 1996.

[18] G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, 1st edition, 1983.

[19] P. C. Saraiva, E. S. Moura, N. Ziviani, B. Ribeiro-Neto, W. Meira, and R. L. C. Fonseca. Rank-preserving two-level caching for scalable search engines. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 51–58, New Orleans, September 2001.

[20] I. Witten, A. Moffat, and T. Bell. *Managing Gigabytes*. Morgan Kaufmann Publishers, New York, second edition, 1999.