A Document-Centric Approach to Static Index Pruning in Text Retrieval Systems

Stefan Büttcher Charles L. A. Clarke

School of Computer Science
University of Waterloo, Canada

{sbuettch,claclark}@plg.uwaterloo.ca

ABSTRACT

We present a static index pruning method, to be used in ad-hoc document retrieval tasks, that follows a document-centric approach to decide whether a posting for a given term should remain in the index or not. The decision is made based on the term's contribution to the document's Kullback-Leibler divergence from the text collection's global language model. Our technique can be used to decrease the size of the index by over 90%, at only a minor decrease in retrieval effectiveness. It thus allows us to make the index small enough to fit entirely into the main memory of a single PC, even for large text collections containing millions of documents. This results in great efficiency gains, superior to those of earlier pruning methods, and an average response time around 20 ms on the GOV2 document collection.

Categories and Subject Descriptors

H.2.4 [**Systems**]: Textual databases; H.3.1 [**Content Analysis and Indexing**]: Indexing methods

General Terms

Experimentation, Performance

Keywords

Information Retrieval, Index Pruning, KL Divergence

1. INTRODUCTION

Fagin et al. [6] introduced the concept of *static index pruning* to information retrieval. In their paper, they describe a *term-centric* pruning method that, for each term T in the index, only retains its top k_T postings, according to the individual score impact that each posting would have if T appeared in an ad-hoc search query (k_T may be term-specific, not necessarily constant). By applying this method to an inverted index, its size can be reduced greatly. At query time, this reduction results in improved query processing performance at the cost of a minor decrease in retrieval effectiveness. Their method is static, as it is applied during index construction and is independent of any search queries.

In contrast to Fagin's method, we present a *document-centric* pruning technique that, for each document D in the text collection, only keeps the postings for the top k_D terms in that document (in general, k_D will not be constant, but will depend on the document D). We use D's Kullback-Leibler divergence from the rest of the collection, and in particular each term's contribution to the document's KL divergence, as a pruning criterion. Conceptually, for every document D in the collection, we perform a pseudo-relevance feedback step, based on Kullback-Leibler divergence scores (described by Carpineto et al. [7]) *at indexing time* and only keep postings for the top k_D feedback terms extracted from that document in the index, discarding everything else. Because pseudo-relevance feedback techniques are very good at finding the set of query terms, given the top search results, this method can be used to very accurately predict the set of queries for which D can make it into the top documents. Only terms appearing in those queries need to be kept in the index.

The resulting pruned index can then either be used standalone or in conjunction with the original, unpruned index. In the latter case, the pruned index acts as the primary index; the unpruned index is used as a secondary index and is only consulted when a query term cannot be found in the pruned index.

By using KLD-based document-centric index pruning to build a pruned index for the 10^6 most frequent terms in the GOV2 [8] text collection, it is possible to construct an index whose size is less than 10% of the size of the original index for the collection, but that still contains most of the information necessary to produce high-quality search results for most search queries. The pruned index is small enough to be loaded into memory and can be used to process the vast majority of all search queries. The original unpruned index, stored on disk because it is too large to fit into main memory, only needs to be accessed for queries involving very rare terms.

Following this general scheme, the average response time of the search engine can be decreased substantially, by about 87% in our experiments (from 190 ms down to 24 ms, when run on a single PC). At the same time, precision at 10 documents only drops by 2.5% (from 0.6400 to 0.6240), and precision at 20 documents by 3.4% (from 0.5660 to 0.5470).

CIKM'06, November 5–11, 2006, Arlington, Virginia, USA.

In the next section, we give a brief overview of related work. This is followed by a general description of the search engine and the baseline retrieval method used in our experiments. In section 4, we present our document-centric approach to index pruning, along with two possible instantiations. Our experimental results are presented in section 5. They include a comparison with existing, term-centric pruning methods and with efficiency figures gathered from a wide range of retrieval systems in the TREC 2005 Terabyte track [8], showing the potential of our new method. Section 6 discusses possible ways to build the global language model needed to compute term scores during index pruning.

Throughout this paper, we use the term *index* to refer to a document-level inverted file (also known as *frequency index*), not containing any positional information. Consequently, *posting* refers to a document-level posting, i.e., a pair of the form (document identifier, term frequency). Indices containing positional information are different in nature and not easily accessible to the pruning techniques described here (de Moura et al. [9] present a possible approach). Our method is not suitable for phrase queries and other query forms that require the existence of positional information in the index. However, it can be used to perform bag-of-words search operations, in conjunction with scoring functions such as Okapi BM25 [13] and certain implementations of language-model-based retrieval functions [2]. If a search query requires access to positional index information, a separate index, containing this information, can be used to process the query. Unless such queries are very frequent, the overall savings achieved by our pruning method can still be substantial. Bahle et al. [3], for instance, report, that only 8.3% of the queries they found in an Excite query log were phrase queries.

2. RELATED WORK

Traditionally, pruning techniques in information retrieval systems have been dynamic, which means that they were applied at query time in order to reduce the computational cost required to find the set of top documents, given a search query. Moffat and Zobel [10], for example, increase their system's query processing performance by restricting the number of document score accumulators maintained in memory in a term-at-a-time query processing framework. Persin et al. [11] describe a dynamic pruning technique that is based on within-document term frequencies. They also show how the inverted index can be reorganized (frequency-sorted index) in order to better support this kind of pruning.

Fagin et al. [6] broke away from the dynamic pruning paradigm and introduced the concept of static index pruning to information retrieval. In static index pruning, information that is not likely to be needed during query processing is discarded during index construction (or immediately following it), resulting in an inverted file that can be much smaller than the original index for the given text collection. The disadvantage of static pruning is that the pruning decision has to be made at indexing time, without any knowledge of the queries that are going to be processed. Its advantage is its ability to substantially reduce the size of the index, promising great efficiency gains due to decreased disk acitivity. In their paper, Fagin et al. describe a global, uniform strategy and several local, term-dependent pruning strategies. They assume that search results are ranked by the search engine based on some TF/IDF method and, during pruning, rank each term's postings according to their hypothetical contribution to a document's TF/IDF score. In their global strategy, every posting whose contribution to the score of the document it belongs to is smaller than a fixed threshold τ is removed from the index. In their term-dependent strategy, $\tau = \tau(T)$ is a function of the term T. Their recommended method lets $\tau(T) = \delta \cdot P_T^{(k)}$, $\delta \in [0, 1]$, where $P_T^{(k)}$ is the score of the kth-best posting for the term T.

Based on Fagin's method, de Moura et al. [9] propose a locality-based pruning technique that, instead of only taking the highest-scoring postings into the pruned index, selects all postings corresponding to terms that appear in the same sentence as one of the postings selected by Fagin's method. Their experimental results indicate that this locality-enhanced version of the pruning algorithm outperforms the original version. They also discuss how pruning techniques can be applied to positional indices in addition to pure frequency indices.

Büttcher and Clarke [5] present a variation of Fagin's method that is motivated by the goal to reduce the size of the index so far that it fits into memory. For very large text collections containing tens of millions of terms (GOV2: around 50 million different terms), this is only possible if the number of terms taken into the pruned index is limited. Therefore, in their pruning strategy, two parameters n and k can be chosen. The pruning process takes an existing index and creates a pruned index that contains the top k postings (using Fagin's technique to sort each term's posting list) for each of the n most frequent terms in the original index. The original index is kept on disk, while the pruned index is loaded into memory. At query time, the search engine fetches postings from the in-memory index whenever possible, accessing the on-disk index only if a query term cannot be found in the pruned in-memory index. Throughout this paper, we refer to this variant of term-centric index pruning as $\mathrm{TCP}_{\mathrm{n}}^{\mathrm{k}}$ (or simply TCP).

In contrast to the *term-centric* pruning methods described above, we propose a *document-centric* strategy, where the decision whether a posting is taken into the pruned index does not depend on the posting's rank within its term's posting list, but on its rank within the document it refers to. In some sense, our technique is similar to the feedback mechanism based on document surrogates that was proposed by Billerbeck and Zobel [4]. The difference is that they use the surrogates for query expansion, while we use them to build the index, and that our term selection method (KL divergence) is different from theirs.

Anh and Moffat [1] have recently presented a very effective pruning technique that is non-static, but requires the index to be in impact-order instead of the traditional document-order. Their technique allows to change the amount of pruning dynamically, after the index has been created, and represents an interesting alternative to our pruning method.

3. INDEX STRUCTURE AND BASELINE RETRIEVAL METHOD

The retrieval system used for the experiments described in this paper is based on a compressed inverted file containing a document-level posting list for each term found in the text collection. Postings are stored as simple integer values, where the 5 least significant bits represent the term frequency (after applying an exponential transformation for

	$k=1$	$k=5$	$k=10$	$k=20$
$f=5$	30.0%	22.4%	18.0%	15.2%
$f=10$	44.0%	35.2%	28.0%	23.6%
$f=20$	64.0%	53.6%	47.2%	39.9%
$f=40$	80.0%	69.6%	64.0%	58.2%

Table 1: Probability that all query terms are among the top f feedback terms of a document randomly chosen from the top k documents retrieved by the BM25 baseline, for various values of f and k.

	$k=1$	$k=5$	$k=10$	$k=20$
$f=5$	94.0%	92.0%	87.6%	79.4%
$f=10$	98.0%	97.6%	96.8%	92.0%
$f=20$	100.0%	99.6%	99.4%	97.8%
$f=40$	100.0%	99.6%	99.8%	99.5%

Table 2: Probability that at least one query term is among the top f feedback terms of a document randomly chosen from the top k documents retrieved by the BM25 baseline, for various values of f and k.

TF values greater than 2^4), while all other bits represent the document number the posting refers to. Postings are grouped into blocks and compressed using a byte-aligned encoding method [14]. Each compressed block contains around 2^{16} postings. After compression, the average size of a posting is 13.6 bits.

Baseline Retrieval Method

Our basic retrieval method relies on the Okapi BM25 formula [13]. Given a search query $\mathcal{Q} = \{Q_1, \ldots, Q_n\}$, containing n terms, the score of a document D is:

$$\mathrm{S}_{\mathrm{BM25}}(D, \mathcal{Q}) = \sum_{i=1}^{n} w_{Q_i} \cdot \frac{f_{D,Q_i} \cdot (k_1 + 1)}{f_{D,Q_i} + k_1 \cdot ((1-b) + b \cdot \frac{dl}{avgdl})},$$

where f_{D,Q_i} is Q_i's frequency within D, dl is D's length (number of tokens), and $avgdl$ is the average document length in the collection. w_{Q_i} is Q_i's IDF weight: $w_{Q_i} = \log(N/N_{Q_i})$, where N is the number of documents in the collection and N_{Q_i} is the number of documents containing the term Q_i. For the free parameters, we chose $k_1 = 1.2$ and $b = 0.5$ – a configuration that was shown to be appropriate for the GOV2 collection used in our experiments [12].

Within this general framework, document scores are computed in a document-at-a-time fashion. For each query term, the corresponding posting list is fetched from the index, decompressed, and the information from all n lists is combined in a multiway merge process, using a heap data structure, resulting in a stream of document scores. Document descriptors for the top documents encountered so far are kept in memory, again using a heap. Standard optimizations, such as MaxScore [15], are applied to reduce the computational cost of processing a query. Additional data structures, not described here, allow us to efficiently look up the score impact of a posting, based on the document number and the term frequency, and to quickly produce official TREC document IDs (e.g., "GX255-91-2697243") from internal document numbers. As a result, queries can be processed with high throughput, at a rate of 5.25 queries per second (190.5 ms per query) for the 50,000 queries used in the TREC 2005 Terabyte efficiency task (with GOV2 as the underlying document collection).

4. DOCUMENT-CENTRIC STATIC INDEX PRUNING

The general goal of our pruning technique is the same as that in [5]: Reducing the size of the index so far that it completely fits into main memory. The difference is that we follow a document-centric approach instead of a term-centric. Our method is similar to the pseudo-relevance feedback mechanism described by Carpineto et al. [7]. Carpineto's method is based on the Kullback-Leibler divergence between the unigram language model defined by an individual document and that defined by the whole text collection. It uses each term's contribution to a document's KL divergence to assign feedback scores to potential expansion terms. Given unigram term distributions P and Q, their KL divergence is:

$$\mathrm{KLD}(P, Q) = \sum_{T \in \mathcal{T}} P(T) \cdot \log\left(\frac{P(T)}{Q(T)}\right), \qquad (1)$$

where $\mathcal{T}$ is the set of all terms in the vocabulary, and $P(T)$ and $Q(T)$ denote T's probability of occurrence under the distribution P and Q, respectively. In their feedback mechanism, Carpineto et al. select a set $\mathcal{R}$ of pseudo-relevant documents, build a language model $\mathcal{M}_R$ for each document $R \in \mathcal{R}$, and compute the feedback score of each term T appearing in $\mathcal{R}$ according to the rule

$$\mathrm{Score}_{\mathrm{FB}}(T) = \sum_{R \in \mathcal{R}} \mathcal{M}_R(T) \cdot \log\left(\frac{\mathcal{M}_R(T)}{\mathcal{M}^*(T)}\right), \qquad (2)$$

where $\mathcal{M}^*$ is the global language model of the entire text collection.

When conducting some initial experiments with this feedback method, we noticed that it is very good at finding the original query terms, given a set $\mathcal{R}$ of pseudo-relevant documents, even if the set is very small ($|\mathcal{R}| = 1$ or $|\mathcal{R}| = 2$). For example, if we pick a random topic from the 50 topics used in the ad-hoc retrieval task of the TREC 2005 Terabyte track and select a single random pseudo-relevant document from the top $k = 20$ documents returned by BM25, then the probability of at least one query term being among the top $f = 20$ feedback terms is 97.8%; the probability of all query terms being among the top $f = 20$ feedback terms is 39.9% (for details, see Tables 1 and 2).

Thus, by performing pseudo-relevance feedback on individual documents in a static fashion *at indexing time*, without taking an actual search query into account, it is possible to predict the set of query terms for which a given document will end up in the top documents returned the BM25 baseline method. Hence, in our document-centric index pruning strategy, a language model $\mathcal{M}_D$ is built for every document D being indexed. Each term T in D is assigned a score:

$$\mathrm{Score}_{\mathrm{DCP}}(T) = \mathcal{M}_D(T) \cdot \log\left(\frac{\mathcal{M}_D(T)}{\mathcal{M}^*(T)}\right). \qquad (3)$$

Only the top-scoring terms are kept in the index; the rest are discarded. The size of the resulting index depends on how many postings for a given document are allowed into the index and how many are rejected. The final goal is to reduce the size of the index enough so that it can be completely loaded into main memory. Since, for a collection with 50 million different terms (GOV2), this is only feasible

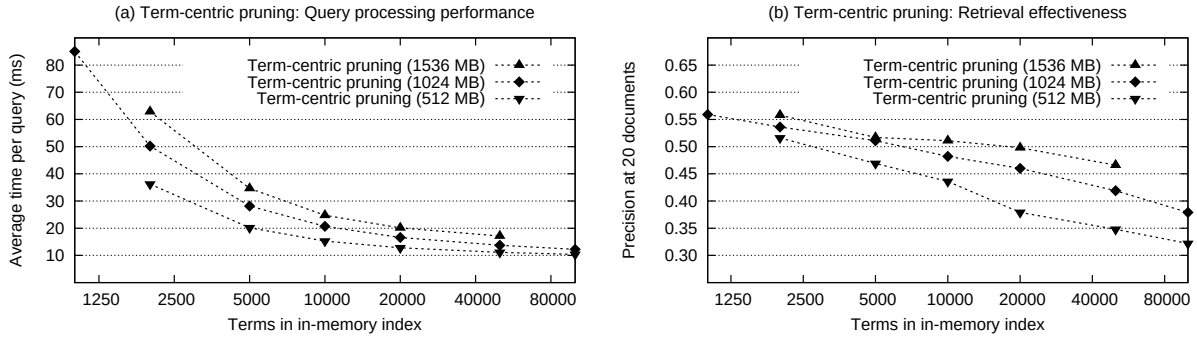

Figure 1: Term-centric pruning (TCP) with 0.5, 1.0, and 1.5 GB of RAM for the pruned in-memory index. Performance gains compared to the retrieval baseline (190 ms) are substantial, but for response times below 20 ms, P@20 decreases considerably in all cases (1536 MB: 12%; 1024 MB: 15%; 512 MB: 17%).

if the number of candidate terms is limited, only the 10^6 most frequent terms in the collection are allowed to enter the pruned index, while the less frequent terms are disregarded completely.

We implemented two different instantiations of the general pruning strategy:

Method 1: $\mathbf{DCP}^{(\mathbf{k})}_{\mathbf{Const}}$

In our first method, for each document in the collection, the top k terms from the document are put into the index; all other terms in the document are discarded. k is a user-defined constant. Smaller k results in a smaller index, while greater k produces higher-quality search results.

Method 2: $\mathbf{DCP}^{(\lambda)}_{\mathbf{Rel}}$

Our second method is similar to the first. Now, however, the number of terms taken from a document D is not constant, but depends on the number of distinct terms in the document, denoted by $|D|$, and a user-defined pruning parameter λ. From each document D, the top $\lceil |D| \cdot \lambda \rceil$ terms are taken into the index; all other terms are discarded. Smaller λ results in a smaller index and lower response times. The rationale behind this method is that longer documents usually cover a greater variety of topics than shorter documents. Thus, the set of possible query terms for longer documents is larger than for shorter ones.

5. EXPERIMENTAL RESULTS

For all our experiments, we used the GOV2 text collection, the result of a webcrawl conducted in 2003/2004, also known as the TREC Terabyte collection [8]. All efficiency figures were obtained by sending the 50,000 efficiency queries used in the TREC 2005 Terabyte track [8] to our search engine and processing all queries in a sequential manner, reporting the top 20 documents for each search query. Since no relevance judgements are available for this query set, all effectiveness numbers were computed for the title-only queries derived from the 50 ad-hoc topics used in the TREC 2005 TB track (topics 751-800). These 50 queries form a subset of the 50,000 efficiency queries. Our evaluation methodology is the same as that used in the Terabyte track: Precision vs. mean time per query. Like in TREC, P@20 (precision after 20 doc's) serves as our main precision measure. In some places, we also look at other measures, such as P@10 and mean average precision (MAP).

All experiments were conducted on a PC based on an AMD Athlon64 3500+ processor (2.2 GHz) with 2 GB of RAM and a 7,200-rpm SATA hard drive. The search engine used in our experiments was a modified version of Wumpus[1]. For the TREC 2005 Terabyte ad-hoc topics, the **baseline retrieval method** presented in section 3 achieves a **P@20 of 0.5660** (P@10: 0.6400; MAP: 0.3346) at an **average response time of 190.5 ms**. Unless explicitly stated otherwise, we always used the pruned in-memory index and the unpruned on-disk index in parallel, fetching postings from the in-memory index whenever possible and only accessing the on-disk index when a query term could not be found in main memory.

In our first series of experiments, we repeated the pruning experiments reported by Büttcher and Clarke [5]. The results shown in Figure 1 are roughly in line with their findings. A slight difference is that, in our experiments, retrieval effectiveness does not drop as quickly as in theirs. This is mainly due to better index compression, which allows us to keep more postings in the in-memory index (Büttcher and Clarke report 19.3 GB for their unpruned index, whereas the unpruned index used in our experiments is only 12.9 GB). This also explains why our baseline (190 ms) is faster than their baseline (326 ms). Despite these improvements, for average reponse times below 20 ms, the difference between TCP and the BM25 baseline is still quite large (for an in-memory index of size 1024 MB, decreasing the response time to under 20 ms requires P@20 to drop below 0.4800 – a 15% decrease compared to the baseline).

We then examined the $\mathrm{DCP}_{\mathrm{Const}}$ pruning strategy. We conducted experiments for various values of the pruning parameter k. Query processing performance is promising, with response times below 25 ms in all cases. Retrieval effectiveness, however, is not even close to that of the BM25 baseline. For $k = 18$, for instance, $\mathrm{DCP}_{\mathrm{Const}}$ achieves an average time per query close to 16.5 ms, but P@20 is 23% below the baseline (0.4370 vs. 0.5660), and MAP, even worse, is 45% below the baseline (0.1835 vs. 0.3346). These numbers are similar to those of term-centric pruning for equivalent query response times (details in Figure 2).

Our experiments with the $\mathrm{DCP}_{\mathrm{Rel}}$ strategy (results shown in Figure 3) had a more reassuring outcome. Like for $\mathrm{DCP}_{\mathrm{Const}}$, query times are very low, less than 25 ms for all values of the pruning parameter λ that we tested.

[1] http://www.wumpus-search.org/

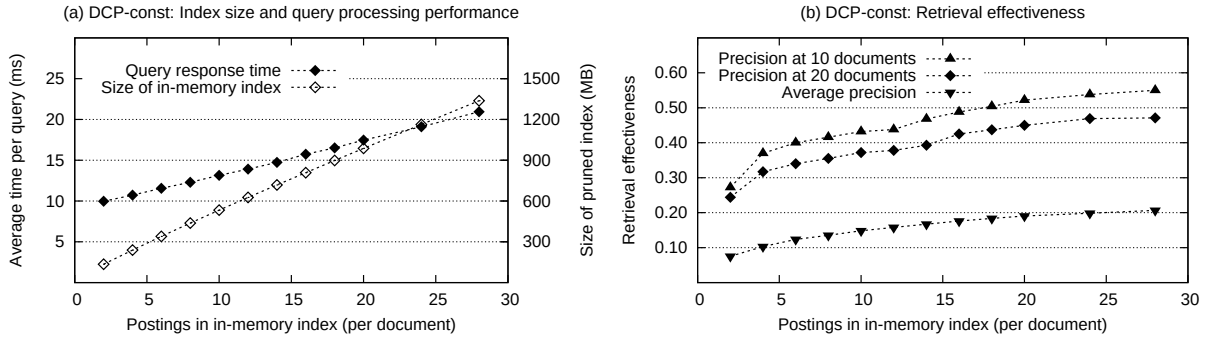

Figure 2: Document-centric pruning with a constant number of postings per document in the in-memory index ($\mathrm{DCP}_{\mathrm{Const}}$). For response times between 10 ms and 21 ms, P@20 varies between 0.2440 ($k = 2$) and 0.4710 ($k = 28$).

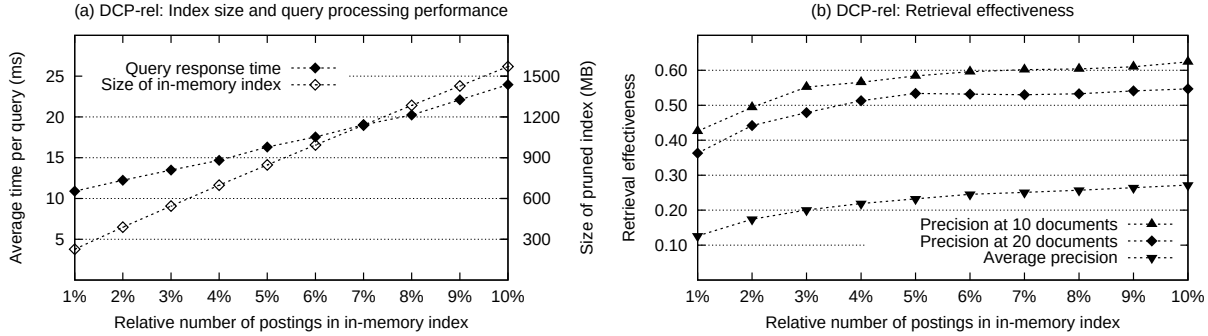

Figure 3: Document-centric pruning with a relative number of postings in the in-memory index ($\mathrm{DCP}_{\mathrm{Rel}}$). For response times between 11 ms and 24 ms, P@20 varies between 0.3630 ($\lambda = 0.01$) and 0.5470 ($\lambda = 0.10$).

Pruning level	Postings decompr.	Postings inspected
$\lambda = 0.04$	242380 (7.5%)	71981 (8.0%)
$\lambda = 0.06$	318273 (9.9%)	94671 (10.6%)
$\lambda = 0.08$	381075 (11.8%)	115862 (13.0%)
$\lambda = 0.10$	442592 (13.7%)	135102 (15.1%)
$\lambda = 1.00$	3219483 (100%)	892145 (100%)

Table 3: Average number of postings touched per query, for $\mathrm{DCP}_{\mathrm{Rel}}$. MaxScore is responsible that the number of postings inspected is not decreasing proportionally to λ.

However, $\mathrm{DCP}_{\mathrm{Rel}}$'s retrieval effectiveness is far superior to that of $\mathrm{DCP}_{\mathrm{Const}}$. With $\lambda = 0.05$ (size of pruned index: 848 MB), $\mathrm{DCP}_{\mathrm{Rel}}$ processes queries at a rate of 61 queries per second (16.3 ms per query). With 0.5340, precision at 20 documents is only 6% below the baseline. Average precision is still rather low (0.2323; 31% below the baseline), but more competitive than that produced by $\mathrm{DCP}_{\mathrm{Const}}$. By increasing λ to 0.08 (resulting in a pruned index of size 1284 MB, 10% the size of the unpruned index), it is possible to improve the situation. Average precision rises to 0.2569, or 23% below the baseline; the average time per query increases to 20 ms.

For $\lambda = 0.1$ (index size: 1570 MB, 12% of unpruned index), the search produced from the pruned index are virtually indistinguishable from those generated from the original, unpruned index. P@20 is 3.4% below the baseline (0.5470 instead of 0.5660) and P@10 only 2.5% (0.6240 instead of 0.6400). With 24 ms per query on average (87% below the baseline), response time is still very low.

The performance gains achieved by $\mathrm{DCP}_{\mathrm{Rel}}$ and documented by our experimental results have two different origins. Firstly, by keeping the primary index in main memory and only accessing the secondary on-disk index when a query term cannot be found in the primary index, disk I/O is decreased dramatically. Secondly, the reduced length of the posting lists in the primary index leads to a smaller number of postings that need to be inspected and thus a smaller number of document scores that need to be computed during query processing.

Table 3 covers this second aspect. It shows that the number of postings inspected during query processing is decreased by up to 92% for $\lambda = 0.04$. The reason why the number of postings that are inspected during query processing is not proportional to the pruning level λ is that the query processor employs the MaxScore [15] heuristic to ignore postings that cannot change the ranking of the top search results (here for the top 20). Thus, many of the postings removed by the pruning process would not have been considered by the query processor anyway. The discrepancy between the number of postings read from the index and decompressed and the number of postings actually used for query processing purposes (a factor-3.5 difference) stems from the fact that our search engine stores postings in compressed blocks of around 2^{16} postings each. Thus, even if only a single posting from a block is needed, the entire block needs to be decompressed. Decreasing the block size might help, but would have other disadvantages, which is why we decided to leave it at 2^{16} postings.

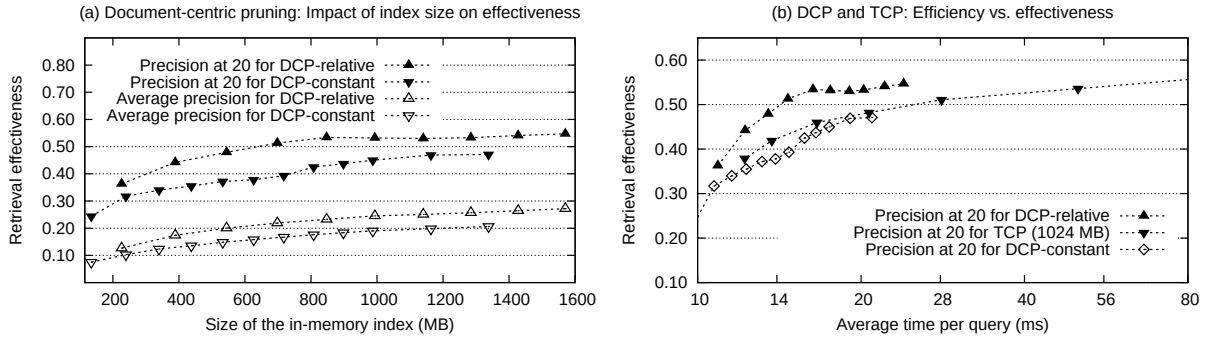

Figure 4: Trade-offs: (a) Space vs. effectiveness and (b) efficiency vs. effectiveness. Even with small amounts of main memory, $\mathrm{DCP}_{\mathrm{Rel}}$ produces good search results and clearly outperforms $\mathrm{TCP}_{\mathrm{n}}^{\mathrm{k}}$.

[TREC 2004 Terabyte queries (topics 701-750)]

	BM25 Baseline	$\mathrm{DCP}_{\mathrm{Rel}}^{(\lambda=0.062)}$	$\mathrm{DCP}_{\mathrm{Const}}^{(\mathrm{k}=21)}$	$\mathrm{TCP}_{(\mathrm{n}=16000)}^{(\mathrm{k}=24500)}$
P@5	0.5224	0.5020	0.4735	0.4490*
P@10	0.5347	0.4837	0.4755	0.4347*
P@20	0.4959	0.4490	0.4224	0.4163
MAP	0.2575	0.1963	0.1621**	0.1808

[TREC 2005 Terabyte queries (topics 751-800)]

	BM25 Baseline	$\mathrm{DCP}_{\mathrm{Rel}}^{(\lambda=0.062)}$	$\mathrm{DCP}_{\mathrm{Const}}^{(\mathrm{k}=21)}$	$\mathrm{TCP}_{(\mathrm{n}=16000)}^{(\mathrm{k}=24500)}$
P@5	0.6840	0.6760	0.6000**	0.5640**
P@10	0.6400	0.5980	0.5300*	0.5380**
P@20	0.5660	0.5310	0.4560**	0.4630**
MAP	0.3346	0.2465	0.1923**	0.2364

Table 4: Comparing $\mathrm{DCP}_{\mathrm{Rel}}$, $\mathrm{DCP}_{\mathrm{Const}}$, and TCP(1024MB) for the title-only queries derived from the TREC ad-hoc topics 701-800. For the same response time (18 ms), $\mathrm{DCP}_{\mathrm{Rel}}$ outperforms the other two strategies at most recall levels.

Figure 4 combines the results we obtained for the 3 different pruning methods and compares their retrieval effectiveness at different efficiency levels. While $\mathrm{DCP}_{\mathrm{Const}}$ performs slightly worse than TCP, the $\mathrm{DCP}_{\mathrm{Rel}}$ pruning strategy outperforms the two other strategies at every efficiency level we tested. At an average response time of 13.5 ms ($\lambda = 0.05$), for example, its P@20 (0.4790) is 14% higher than that of TCP (0.4190) and 27% higher than that of $\mathrm{DCP}_{\mathrm{Const}}$ (0.3780). In addition, Figure 4(a) shows that even for a relatively small in-memory index (about 700 MB), $\mathrm{DCP}_{\mathrm{Rel}}$ achieves respectable precision (P@20 = 0.5130).

Statistical Significance

We fixed the amount of main memory available for the in-memory index to 1024 MB. With an in-memory index of this size, $\mathrm{DCP}_{\mathrm{Const}}$ ($k = 21$) and $\mathrm{DCP}_{\mathrm{Rel}}$ ($\lambda = 0.062$) both lead to an average query response time of 18 ms for the TREC 2005 Terabyte efficiency queries. With TCP, the same response time can be achieved by building an index containing the $k = 24500$ best postings for each of the $n = 16000$ most frequent terms ($\mathrm{TCP}_{16000}^{24500}$). We analyzed all three pruning techniques at this efficiency level, using the ad-hoc topics from the 2004 and 2005 TREC Terabyte tracks as test queries. Table 4 provides precision values at several retrieval points for each pruning method. It shows that $\mathrm{DCP}_{\mathrm{Rel}}$ declasses $\mathrm{DCP}_{\mathrm{Const}}$ and TCP at virtually every recall level. Stars indicate significantly worse retrieval effectiveness for $\mathrm{DCP}_{\mathrm{Const}}$ and TCP, compared with $\mathrm{DCP}_{\mathrm{Rel}}$, according to a paired t-test (one star: 95% confidence; two stars: 99% confidence).

Similarity to Original Search Results

Another interesting question is how close the search results produced by $\mathrm{DCP}_{\mathrm{Rel}}$ are to those produced by the BM25 baseline – in other words, to find out whether $\mathrm{DCP}_{\mathrm{Rel}}$ achieves such high precision because it actually produces the similar search results as the baseline, or because it returns different documents which, however, also turn out to be relevant. To evaluate the similarity between the baseline results and the $\mathrm{DCP}_{\mathrm{Rel}}$ results, we employ the methodology also used by Fagin et al. [6]. Table 5 shows that, for the TREC 2005 Terabyte ad-hoc topics, the search results produced from the pruned indices are in fact very similar to those produced from the unpruned index. For $\lambda = 0.1$, the similarity level, as measured by the symmetrical difference between the search results, i.e., the ratio of the intersection and the union of the top 20 documents produced from the pruned and from the unpruned index, is 67%. On average, 77% of the top 20 doc's produced from the unpruned index appear in the top 20 doc's produced from the pruned index.

TREC Terabyte

We compared the performance of our pruning method to other retrieval systems that participated in the efficiency task of the TREC 2005 Terabyte track (as reported by Clarke et al. [8]). Figure 5 shows that document-centric

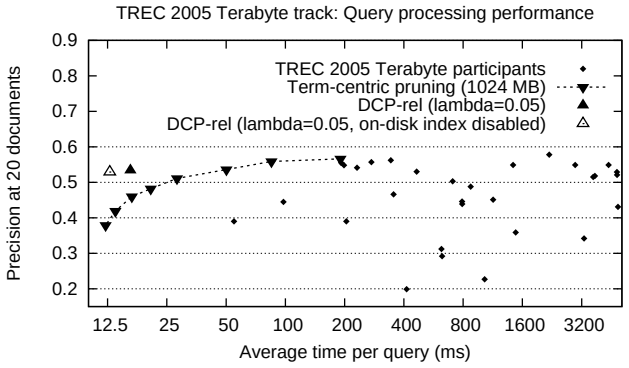

Figure 5: Comparing term-centric and document-centric index pruning with the official runs submitted for the TREC 2005 Terabyte track. All query times are CPU-normalized – simulating search engines running on a single-CPU system.

pruning allows us to protrude into the previously undiscovered area in the top-left corner of the graph, demonstrating the potential of DCP. The fastest run submitted for the efficiency task of the Terabyte track had a mean time per query of 55 ms (normalized by #CPUs) and P@20 = 0.3900. At that precision level, $\mathrm{DCP}_{\mathrm{Rel}}$ needs less that 12 ms per query. However, the evaluation methodology employed in the Terabyte track (response times normalized by #CPUs) is questionable, and the inter-system comparison should be taken with a grain of salt.

Additional Experiments

For all experiments described so far, the pruned in-memory index and the unpruned on-disk index were used in parallel. The unpruned index served as a backup that could be accessed whenever no postings for a given query term were found in the pruned index. In experiments not reported on in detail here, we examined how turning off the on-disk index affects both efficiency and effectiveness. For a $\mathrm{DCP}_{\mathrm{Rel}}$-pruned index with $\lambda = 0.05$, average response time can be decreased from 16.3 ms to 12.8 ms by disabling the on-disk index. On the other hand, this makes P@20 drop from 0.5340 to 0.5290 because "mersenne" (topic 784) and "geechee" (topic 791) have no posting list in the pruned index. This confirms our initial assumption that a backup index might be needed to cover the case of query terms with very low collection frequency.

By looking at the pruned index created by $\mathrm{DCP}_{\mathrm{Rel}}^{(\lambda=0.03)}$, we noticed that, although it was only 544 MB large (4.1% of the unpruned index), the posting list for the term "the" had 3.0 million entries (15% of its length in the unpruned index). Our KLD-based pruning techniques seems to favor frequent terms over unfrequent terms. By changing the term score formula from equation 3 to

$$(\mathcal{M}_D(T))^{1-\delta} \cdot \left(\max \left\{ 0,\ \log \left(\frac{\mathcal{M}_D(T)}{\mathcal{M}^*(T)} \right) \right\} \right)^{1+\delta}, \qquad (4)$$

with $\delta \in [0, 1)$, it is possible to counter this effect. Choosing $\delta = 0.10$, for example, cuts the list for "the" down to 2.37 million entries. It increases P@10 from 0.5520 to 0.5580 and P@20 from 0.4790 to 0.4890, without harming performance. We did not further investigate in this direction.

Pruning level	1 - symm.diff.	Kendall's τ (top-20)
$\lambda = 0.04$	0.4403	0.6915
$\lambda = 0.06$	0.5421	0.7753
$\lambda = 0.08$	0.6090	0.8197
$\lambda = 0.10$	0.6716	0.8557

Table 5: Similarity of top 20 search results produced from pruned index to top 20 search results from unpruned index, for $\mathrm{DCP}_{\mathrm{Rel}}$ and various pruning levels λ. Topics: TREC Terabyte 2005.

Finally, we compared our KLD-based term selection function to the formula proposed by Billerbeck and Zobel [4]. Although Billerbeck's objective (query expansion) is different from ours (index pruning), the techniques are similar, raising the question how their method performs compared to ours. We modified the $\mathrm{DCP}_{\mathrm{Rel}}^{(\lambda)}$ strategy so that it uses the selection function

$$\mathrm{Score}_{\mathrm{DCP}}(T) = \log(N/N_T) \cdot \log(f_{D,T} + 1) \qquad (5)$$

instead of equation 3. Here, N is the total number of documents in the collection, N_T the number of documents containing T, and $f_{D,T}$ the frequency of the term T within the document in question. For equivalent query processing performance (17.5 ms per query), Billerbeck's selection function produces a P@20 of 0.4820 — 9.4% lower than the original KLD-based selection function (0.5320). A paired t-test reports statistical significance with confidence level 94.3%.

6. BUILDING THE LANGUAGE MODEL

The pruning method described in section 4 demands the knowledge of the global language model of the text collection for which an index is being constructed. This seems to require a two-pass index construction process, where the language model is built during the first phase and the actual index is constructed during the second phase, approximately doubling the total index construction time. Fortunately, if the text collection is very large, it is not necessary to analyze the entire collection during the first phase. A representative subcollection, consisting of randomly selected documents, can be used to build the language model in phase 1, and the whole collection does not need to be read twice.

For all experiments described in the previous section, we used a language model built from a random subcollection comprising 5% of the documents in the GOV2 corpus. Altering the size of the subcollection does not substantially change the effectiveness of our technique. For $\mathrm{DCP}_{\mathrm{Rel}}^{(0.062)}$ (index size: 1024 MB), lowering the size of the subcollection from 5% to 1% results in a mild decrease of P@20 (from 0.5310 to 0.5250). Increasing it to 10%, improves P@20 to 0.5340. We also conducted experiments in which we used a completely different text collection to build the background language model. If TREC disk 5 is used to build the background model, P@20 decreases to 0.5270. If the Medline corpus from the TREC Genomics track is used, P@20 drops to 0.5070. Since the TREC Medline corpus is as far away from GOV2 as it can get, without changing the language of the collection from English to something else, we conclude that our method is very robust with respect to noisy data in the language model.

Another aspect of our approach is data sparseness. We decided to build the language model according to the max-

imum likelihood estimate. MLE is very amenable to inaccuracies resulting from data sparseness. We addressed this problem by restricting the language to the X most frequent terms seen in the subcollection (in our experiments: $X = 10^6$). This requires us to build an unpruned index in addition to the pruned index so that, at query time, posting lists for infrequent terms, which did not make it into the pruned index, can be fetched from the unpruned index. However, this does not imply the necessity of a two-phase index construction process. Instead, two indices can be built in a single pass – the full index for the collection, containing all postings, and a smaller, pruned index that may be loaded into memory for query processing.

7. CONCLUSION

We have presented a document-centric index pruning method that can be used to dramatically increase the query processing performance of search engines working on static document collections. Our pruning method employs Kullback-Leibler divergence to select terms by performing query-independent pseudo-relevance feedback at indexing time. We examined two possible implementations of our document-centric pruning method. The first implementation selects a constant number of terms from each document, while the number of terms selected by the second implementation is proportional to the number of distinct terms in the document. Our experimental results show that the proportional pruning method (DCP_{Rel}), outperforms existing term-centric pruning methods.

Compared to the Okapi BM25 baseline retrieval method implemented in our search engine, DCP_{Rel} can decrease the average response time by 87% (to 24 ms) for the efficiency queries used in the TREC Terabyte track. At the same time, P@20 only decreases by 3.4% (from 0.5660 to 0.5470). Further speed-ups are possible, but carry the cost of a larger decrease of the search engine's effectiveness: Decreasing the response time by 92.3% (to 14.7 ms) makes P@20 drop by 9.4%, to 0.5130. Compared to term-centric pruning methods, document-centric pruning also has the advantage that the pruning criterion can be chosen independently of the ranking function employed for query processing; there is no obvious, close connection between KL divergence and the incarnation of BM25 we use to rank the search results.

The main part of the savings achieved by our method stems from the decrease (or elimination) of disk transfer operations, which represent a major bottleneck for most retrieval systems. In distributed search systems that consist of a number of computers large enough to hold the entire unpruned index in memory, the efficiency gains will not be as dramatic as indicated by our experiments. However, even in such cases, our pruning method can be expected to improve query efficiency by a factor 5 or so, by reducing the amount of postings that need to be inspected during query processing.

8. REFERENCES

[1] V. N. Anh and A. Moffat. Pruned Query Evaluation Using Precomputed Impacts. In *Proceedings of the 29th ACM SIGIR Conf. on Research and Development in Information Retrieval*, Seattle, USA, 2006.

[2] L. Azzopardi and D. E. Losada. An Efficient Computation of the Multiple-Bernoulli Language Model. In *Proceedings of the 28th European Conference on Information Retrieval (ECIR 2006)*, pages 480–483, London, UK, April 2006.

[3] D. Bahle, H. E. Williams, and J. Zobel. Efficient Phrase Querying with an Auxiliary Index. In *Proceedings of the 25th ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 215–221, Tampere, Finland, 2002.

[4] B. Billerbeck and J. Zobel. Techniques for Efficient Query Expansion. In *Proceedings of the 11th Symposium on String Processing and Information Retrieval*, pages 30–42, Padova, Italy, September 2004.

[5] S. Büttcher and C. L. A. Clarke. Efficiency vs. Effectiveness in Terabyte-Scale Information Retrieval. In *Proceedings of the 14th Text REtrieval Conference*, Gaithersburg, USA, November 2005.

[6] D. Carmel, D. Cohen, R. Fagin, E. Farchi, M. Herscovici, Y. Maarek, and A. Soffer. Static Index Pruning for Information Retrieval Systems. In *Proceedings of the 24th ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 43–50, 2001.

[7] C. Carpineto, R. de Mori, G. Romano, and B. Bigi. An Information-Theoretic Approach to Automatic Query Expansion. *ACM Transactions on Information Systems*, 19(1):1–27, 2001.

[8] C. Clarke, F. Scholer, and I. Soboroff. Overview of the TREC Terabyte Track. In *Proceedings of the 14th Text REtrieval Conference*, Gaithersburg, USA, November 2005.

[9] E. S. de Moura, C. F. dos Santos, D. R. Fernandes, A. S. Silva, P. Calado, and M. A. Nascimento. Improving Web Search Efficiency via a Locality Based Static Pruning Method. In *Proceedings of the 14th International Conference on World Wide Web*, pages 235–244, New York, USA, 2005.

[10] A. Moffat and J. Zobel. Self-Indexing Inverted Files for Fast Text Retrieval. *ACM Transactions on Information Systems*, 14(4):349–379, October 1996.

[11] M. Persin, J. Zobel, and R. Sacks-Davis. Filtered Document Retrieval with Frequency-Sorted Indexes. *Journal of the American Society for Information Science*, 47(10):749–764, October 1996.

[12] V. Plachouras, B. He, and I. Ounis. University of Glasgow at TREC2004: Experiments in Web, Robust and Terabyte Tracks with Terrier. In *Proceedings of the 13th Text REtrieval Conference*, Gaithersburg, USA, November 2004.

[13] S. E. Robertson, S. Walker, and M. Hancock-Beaulieu. Okapi at TREC-7. In *Proceedings of the Seventh Text REtrieval Conference*, Gaithersburg, USA, November 1998.

[14] F. Scholer, H. E. Williams, J. Yiannis, and J. Zobel. Compression of Inverted Indexes for Fast Query Evaluation. In *Proceedings of the 25th ACM SIGIR Conference on Research and Development in Information Retrieval*, Tampere, Finland, August 2002.

[15] H. Turtle and J. Flood. Query Evaluation: Strategies and Optimization. *Information Processing & Management*, 31(1):831–850, November 1995.