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

Stefan Büttcher  Charles L. A. Clarke

University of Waterloo, Canada

November 7, 2006
Static Index Pruning
General Idea

- An inverted index for a text collection often contains data that are never used during query processing.

- We want to predict which postings (i.e., term/document pairs) will be needed to produce the search results and which will not.

- We restrict ourselves to the situation in which the top 20 documents for a given search query have to be found.

- Removing postings from the index that are unlikely to be used during query processing can improve the search engine’s response time.

- If we remove the right postings, precision is not harmed (hopefully...).
If the pruned index could be small enough to fit into main memory, then this would be really good.

Unfortunately, the number of distinct terms is too large. GOV2: 50 million terms. $50M \times 20 \text{ bytes} = 1\text{GB}$.

Solution: Build a pruned index for the frequent terms, e.g. top 1M terms. Load the pruned index into main memory.

Whenever a term cannot be found in the pruned in-memory index, consult the unpruned on-disk index.
Term-Centric Index Pruning
Carmel et al. (2001):

- Choose a scoring function, e.g., Okapi BM25.
- Pick an integer $k$ and some $\varepsilon$ from $[0, 1]$.
- For each term $T$ in the index, compute the score impact of the $k$-th best posting, according to the selected scoring function: $S_T^{(k)}$.
- Set the term-specific pruning threshold to $\theta_T := \varepsilon \times S_T^{(k)}$.
- All postings with impact smaller than $\theta_T$ are removed from the index.
A Variation

Büttcher and Clarke (2005):

- Very similar to Carmel’s method.
- Pick an integer $N$.
- For each of the $N$ most frequent terms, keep the top $k$ postings and discard the rest.
- $N$ and $k$ are chosen in such a way that the resulting index fits into main memory.
- We made good experience with this method in TREC Terabyte 2005.

In our experiments, we exclusively used this variant of Carmel’s method.
Pseudo-Relevance Feedback
Given two probability distributions $P$ and $Q$, the Kullback-Leibler divergence between $P$ and $Q$ is:

$$\text{KLD}(P, Q) = \sum_x P(x) \cdot \log \left( \frac{P(x)}{Q(x)} \right).$$

KL divergence is also referred to as relative entropy.

If $P$ and $Q$ are unigram language models, $P$ generated from a document $D$, and $Q$ generated from the whole text collection, then can use $\text{KLD}(P, Q)$ as an indicator for how different $D$ is from the rest of the collection.
Carpineto et al. (2001) use Kullback-Leibler Divergence to select query expansion terms through pseudo-relevance feedback:

- Intuition: A good expansion term is a term that distinguishes a relevant document from the rest of the collection.
- Let $\mathcal{M}_C$ be the unigram language model for the text collection.
- Let $\mathcal{M}_D$ be the unigram language model for the pseudo-relevant document $D$.
- Assign each term $T$ in $D$ a score:
  $$S_D(T) = \mathcal{M}_D(T) \cdot \log \left( \frac{\mathcal{M}_D(T)}{\mathcal{M}_C(T)} \right).$$
- Pick top $f$ terms as expansion terms, according to their score.
Experiments with KLD-Based Feedback

Probability that *all query terms* are among the top $f$ feedback terms, for top $k$ documents retrieved by BM25:

<table>
<thead>
<tr>
<th></th>
<th>$k = 1$</th>
<th>$k = 5$</th>
<th>$k = 10$</th>
<th>$k = 20$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f = 5$</td>
<td>30.0%</td>
<td>22.4%</td>
<td>18.0%</td>
<td>15.2%</td>
</tr>
<tr>
<td>$f = 10$</td>
<td>44.0%</td>
<td>35.2%</td>
<td>28.0%</td>
<td>23.6%</td>
</tr>
<tr>
<td>$f = 20$</td>
<td>64.0%</td>
<td>53.6%</td>
<td>47.2%</td>
<td>39.9%</td>
</tr>
<tr>
<td>$f = 40$</td>
<td>80.0%</td>
<td>69.6%</td>
<td>64.0%</td>
<td>58.2%</td>
</tr>
</tbody>
</table>

(for 50 ad-hoc topics from TREC Terabyte 2005)
Experiments with KLD-Based Feedback

Probability that at least one query term is among the top $f$ feedback terms, for top $k$ documents retrieved by BM25:

<table>
<thead>
<tr>
<th>$f$</th>
<th>$k = 1$</th>
<th>$k = 5$</th>
<th>$k = 10$</th>
<th>$k = 20$</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>94.0%</td>
<td>92.0%</td>
<td>87.6%</td>
<td>79.4%</td>
</tr>
<tr>
<td>10</td>
<td>98.0%</td>
<td>97.6%</td>
<td>96.8%</td>
<td>92.0%</td>
</tr>
<tr>
<td>20</td>
<td>100.0%</td>
<td>99.6%</td>
<td>99.4%</td>
<td>97.8%</td>
</tr>
<tr>
<td>40</td>
<td>100.0%</td>
<td>99.6%</td>
<td>99.8%</td>
<td>99.5%</td>
</tr>
</tbody>
</table>

(for 50 ad-hoc topics from TREC Terabyte 2005)
Document-Centric Index Pruning
Query-Independent Feedback

- Document-centric index pruning uses KLD-based pseudo-relevance feedback.
- Prune the index by performing query-independent pseudo-relevance feedback on individual documents.
- From each document, keep the top terms according to feedback score. Discard the rest.
- How many terms per document should be kept?
The simplest selection strategy is referred to as $\text{DCP}_{\text{const}}$:

- Choose an integer $k$.
- From each document, keep the top $k$ feedback terms.
- Discard the rest.
DCP\textsubscript{rel} is unfair against longer and more diverse documents.

Longer documents may cover a greater variety of topics and should therefore be allowed to contribute a greater number of terms to the pruned index.

This is realized by DCP\textsubscript{rel}:

- Choose a parameter $\lambda$ from $[0, 1]$.
- From each document $D$ keep the top $|D| \times \lambda$ terms, where $|D|$ denotes the number of distinct terms in $D$.
- Discard the rest.
Experimental Results
Experimental Setup

Index pruning is evaluated by measuring different efficiency/effectiveness trade-off points.

- Text collection: GOV2 (25.2 million documents, 44 billion tokens).
- Efficiency: 50,000 efficiency topics from TREC TB 2005.
- Retrieval baseline: BM25 with a document-level frequency index (stored on disk). P@20 = 0.5660. Average time per query: 190.5 ms.
Term-Centric Pruning

(a) Term-centric pruning: Query processing performance

(b) Term-centric pruning: Retrieval effectiveness
DCP\textsubscript{const}

(a) DCP-const: Index size and query processing performance

- Query response time
- Size of in-memory index

Average time per query (ms)

Size of pruned index (MB)

Postings in in-memory index (per document)
DCP_{\text{const}}

(b) DCP-const: Retrieval effectiveness

- Precision at 10 documents
- Precision at 20 documents
- Average precision

Retrieval effectiveness vs. Postings in in-memory index (per document)

Precision at 10 documents:
- Initial value: 0.10
- Increase with postings

Precision at 20 documents:
- Initial value: 0.20
- Increase with postings

Average precision:
- Initial value: 0.30
- Increase with postings

Stefan Büttcher · University of Waterloo
(a) DCP-rel: Index size and query processing performance

- **Query response time**
- **Size of in-memory index**

- **Average time per query (ms)**
- **Size of pruned index (MB)**

- **Relative number of postings in in-memory index**
Experimental Setup

Document-Centric Pruning

Efficiency vs. Effectiveness

Statistical Significance

Similarity to Original Search Results

DCP$_{rel}$

(b) DCP-rel: Retrieval effectiveness

Precision at 10 documents

Precision at 20 documents

Average precision
Efficiency vs. Effectiveness

TREC 2005 Terabyte track: Query processing performance

- TREC 2005 Terabyte participants
- Term-centric pruning (1024 MB)
- DCP-rel (lambda=0.05)
- DCP-rel (lambda=0.05, on-disk index disabled)

Precision at 20 documents vs. Average time per query (ms)
## Statistical Significance

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

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

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

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

*: $p < 0.05$ (compared to $\text{DCP}_{\text{rel}}$)  
**: $p < 0.01$ (compared to $\text{DCP}_{\text{rel}}$)
Similarity to Original Search Results

Comparing the search results produced from the pruned index with the results produced from the unpruned index.

- Symmetrical difference: \( 1 - \frac{|\text{intersection}|}{|\text{union}|} \) (top 20 documents).
- Kendall’s \( \tau \): Modified version, restricted to top-20 results.

<table>
<thead>
<tr>
<th>Pruning level</th>
<th>1 - symm.diff.</th>
<th>Kendall’s ( \tau ) (top-20)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \lambda = 0.04 )</td>
<td>0.4403</td>
<td>0.6915</td>
</tr>
<tr>
<td>( \lambda = 0.06 )</td>
<td>0.5421</td>
<td>0.7753</td>
</tr>
<tr>
<td>( \lambda = 0.08 )</td>
<td>0.6090</td>
<td>0.8197</td>
</tr>
<tr>
<td>( \lambda = 0.10 )</td>
<td>0.6716</td>
<td>0.8557</td>
</tr>
</tbody>
</table>
Document-centric index pruning is based on query-independent per-document pseudo-relevance feedback.

Document-centric index pruning can be used to substantially decrease the average time per query (from 190 ms to 20 ms in our experiments).

If not applied too aggressively, search quality (P@20) is almost not harmed.

Search results stay reasonably close to the original results produced from the unpruned index.

In a search engine, index pruning can be enabled selectively, to overcome short periods of very high query activity.
The End

Thank You!