Document Compaction for Efficient Query Biased Snippet Generation

Yohannes Tsegay[1], Simon J. Puglisi[1], Andrew Turpin[1], and Justin Zobel[2]

[1] School of Computer Science and IT, RMIT University, Melbourne, Australia
[2] Dept. Computer Science and Software Engineering,
University of Melbourne, Australia

Abstract. Current web search engines return query-biased snippets for each document they list in a result set. For efficiency, search engines operating on large collections need to cache snippets for common queries, and to cache documents to allow fast generation of snippets for uncached queries. To improve the hit rate on a document cache during snippet generation, we propose and evaluate several schemes for reducing document size, hence increasing the number of documents in the cache. In particular, we argue against further improvements to document compression, and argue for schemes that prune documents based on the *a priori* likelihood that a sentence will be used as part of a snippet for a given document. Our experiments show that if documents are reduced to less than half their original size, 80% of snippets generated are identical to those generated from the original documents. Moreover, as the pruned, compressed surrogates are smaller, 3-4 times as many documents can be cached.

1 Introduction

To assist users in identifying documents relevant to their information need, typical web search engines provide a brief summary – or *snippet* – of each document on the results page. Snippets most often consist of two or three sentences (or fragments) extracted from the retrieved document or its metadata. In early retrieval systems, snippets were *generic*, consisting of some predetermined portion of each document – the first b words for example. Today, the major search engines provide *query-biased* snippets that consist of parts of the document (sentences or sentence fragments) that are in some way pertinent to the query.

It is perhaps not surprising that users prefer query-biased snippets over generic [13]: by showing the user how the query terms are used in the context of a document, query-biased snippets reduce the need to refer to the full document. However, this quality comes at the cost of increased processing load. Because the search engine does not know *a priori* the set of queries for which a document may be fetched, it must retain each document in some form, to be searched and processed with respect to a query each time the document is highly ranked.

Turpin et al. [14] studied the computation involved in snippet generation, finding that 70%–80% of snippet generation time is spent fetching a document

M. Boughanem et al. (Eds.): ECIR 2009, LNCS 5478, pp. 509–520, 2009.

from disk, with the remaining time spent searching and scoring sentences relative to query terms in main memory. This is a substantial component of retrieval cost as, for every query, ten or more documents must be fetched from disk and summarized. Turpin et al. [14] show that explicit caching of documents in main memory is a highly effective way to increase querying throughput.

In this paper we propose new document compaction schemes with the aim of further speeding query-biased snippet generation. The potential effect of compaction on processing throughput is twofold. First, the number of costly disk seeks and reads is reduced as compaction allows more documents to be stored in cache. Second, there are fewer sentences to evaluate during the scoring phase, so processing times are also reduced.

Our approach to document compaction begins by reordering the sentences in a document so that those more likely to be included in a query-biased snippet appear near the start of the document. Once reordered, the document is pruned to the desired length by discarding all but the highest ranking sentences.

When generating snippets, the pruned documents are used as surrogates of the full. Our experimental results show that, using these surrogates, 60%–80% of queries produce snippets that are identical to those generated using full-documents. In some situations it is possible to detect when pruned documents are producing inferior snippets, in which case snippet generation can defer to the unpruned document available on disk. We argue that such an approach is superior to caching compressed, original documents. These findings are supported by experiments run on several large TREC web collections of 10 GB and 100 GB.

2 Related Work

Snippets are a form of extractive document summaries. The use of automatic extractive document summarisation dates back to 1950s, when Luhn [9] proposed that a summary should be composed of the most significant sentences in a document; significant sentences contain clusters of significant terms, and a term is considered significant based on its frequency in the document. Similar sentence selection principles have since been the general theme in much of the work in document summarisation in text information retrieval systems [6,10,11,13]. For summaries presented in search result list captions, Tombros and Sanderson study the advantages of *query-biased summaries* [13], in which sentence fragments that best match the query are selected. Other sentence features (such as presence in heading or position in document) are also used to select sentences that not only match the query but may also indicate the topic of the document. Tombros and Sanderson found that users spent less time searching and performing post-search relevance judgments when presented with query-biased summaries compared to generic summaries. In a later study, White et al. [15] confirmed user preference for query-biased summaries, this time by measuring the time it took users to complete search tasks.

Despite the utility of query-biased summaries for web search, surprisingly little published work addresses methods for generating them. Silber and McCoy [12]

propose an efficient method of generating an intermediate document representation that can then be used to formulate summaries. The focus of their work, however, is not the efficient generation of summaries, but rather the efficient generation of the intermediate document representations.

Turpin et al. [14] propose a compressed token system (CTS) for efficient snippet generation. Using document compression, and in-memory caching of documents, they demonstrate that on average the time it takes to generate snippets can be reduced to a fifth of the obvious baseline. However the document representation they propose makes use of query logs, which poses two main problems. First, if the nature of queries drifts the document representation must also change, adding computational overhead. Second, the presence of a suitable query log is assumed. These shortcomings highlight the need for an adaptive compaction scheme that can function independent of past queries, which is our aim in this work.

Several authors have utilized document compaction or pruning schemes for information retrieval tasks other than snippet generation. De Moura et al. [5] use document pruning to retain only the "novel" sentences in a document, with the pruned documents then used to construct a smaller (pruned) inverted index which supports phrase queries. Lu and Callan [8] propose an approach that selects and retains keywords in a document in order to reduce the size of sample documents in a distributed retrieval environment. Billerbeck and Zobel [2] use a keyword-based document pruning approach to construct document surrogates for efficient query expansion.

Alternatively, document size can be reduced by *lossless* means, that is, by compression. Using semi-static compression approaches, prior work shows that a large text collection can be compressed down to 30% of its uncompressed size [14]. However, as we argue below, compression has costs as well as benefits.

3 Against Further Compression

In experiments with snippet creation, Turpin et al. [14] found that 68% of snippet generation time is spent fetching documents from disk if a document to be summarized is not stored in a document cache. It is valuable, therefore, for as many documents as possible to be in the search engine's document cache, increasing the chances that a request for a document from a snippet generation algorithm hits that cache and does not go to disk. Assuming a fixed amount of memory is available for such a cache, any reduction in document size can lead to a higher number of documents within the cache.

There are two possible approaches to reducing document size: lossless compression, or lossy compaction. Turpin et al. [14] introduced a compression scheme that was practical for large collections, and worked well in their experiments. But is it worth investing effort in improving such a compression scheme? First, define the average time it takes to generate a snippet for a document using the existing scheme as

$$T = \alpha F_c + (1 - \alpha) F_d + C + S,$$

where $0 \leq \alpha \leq 1$ is the cache hit-rate, F_c and F_d are the average time to fetch a compressed document from cache and disk respectively, C is the average CPU time to decompress a document, and S is the average time to score and rank sentences in a document.

Assuming that any improvement in compression ratios would come at the cost of increased decoding time, a new compression scheme would require average snippet generation time per document of

$$T' = \beta F_c + (1 - \beta)F_d + (1 + r)C + S,$$

where $\alpha \leq \beta \leq 1$ is the new cache hit-rate achieved because documents are smaller, and the new average decode time is a factor of $0 \leq r < F_d/C - 1$ slower than the old decode time C. Note that we are assuming that the average decode time will not increase to F_d, the average time required to fetch a document from disk, because then any advantage of caching the document is lost.

For the new scheme to be faster, $T' \leq T$, which implies that $\beta \geq \alpha + rC/(F_d - F_c)$. From this equation, when the current hit-rate is $\alpha = 10\%$, then a new compression scheme can be 50% slower than the existing scheme ($r = 0.5$), as long as the cache hit-rate is improved to at least $\beta = 33\%$. From the simulations reported in Figure 4 of Turpin et al. [14], improving the hit-rate from 10% to 33% requires a very small increase in the amount of the collection in the cache, hence a small increase in compression ratios, so may be achievable. Using results from the Canterbury Corpus (`corpus.canterbury.ac.nz`), the bzip2 compression scheme is about 1.5 the speed of the huffword2 scheme, which is similar to that used by Turpin et al., and compresses to data to about half the size of that of huffword2, so would be a suitable choice.

Figure 4 of Turpin et al. shows that gains in snippet generation speed are non-linear with cache size. Hence increasing the size of a small cache leads to larger speed ups than increasing an already large cache. With only 1% of all documents in cache, α is more likely to be around 80%. Assuming a small increase in decode speed by a factor of $r = 20\%$ requires the new cache hit-rate to be $\beta = 89\%$. From the previous caching simulations, this would require about 3% of the collection to be cached: hence, the new compression scheme would need to compress documents to a third of their current compressed size with only a 20% increase in decode time. This seems unlikely to be possible.

In summary, therefore, document cache hit-rates are quite high with simple, fast compression schemes, but there are diminishing returns in investing more effort in compression schemes. As the hit-rate gets higher, it is increasingly difficult to trade a decrease in compression speed for reduced document size. Given that improved compression schemes are unlikely to decrease average snippet generation time, we concentrate on lossy compaction techniques as a method for getting more documents into cache. In the next section we introduce our document compaction approach based on sentence reordering and then pruning reordered documents.

4 Document Compaction

To generate compact surrogate documents, we need to first reorder sentences then prune sentences that are less likely to contribute to surrogates. We now describe our approaches to sentence reordering and pruning.

Sentence reordering. To compact a document we begin by assigning each sentence in the document a weight and then reorder the sentences in descending order of weight. All but the top N sentences are then removed, with N specifying the desired level of compaction. Sentence weights are a linear combination of the weights of the individual terms that comprise the sentence. A term's weight is a numerical value that reflects its importance or impact in that document. We investigate two separate term weighting approaches used in existing text retrieval models, namely TFIDF [1,18] and Kullback-Leibler divergence [4,7].

TF·IDF weighting. This is similar to Luhn's concept of *term significance* [9]. However, in addition to the term's frequency in a document (TF), the inverse document frequency (IDF) – reciprocal of the count of documents which contain the term – is also incorporated to establish term importance. By including the IDF component, terms that occur frequently in a document but are rare across the collection are given higher weight. The weight of a term t is computed as a product of TF and IDF components. Here, we use a combination of TF and IDF as specified by Zobel and Moffat [17]:

$$\text{TF} = 1 + \ln(f_{d,t}),\ \text{IDF} = \ln\left(\frac{N}{d_f}\right),$$

where $f_{d,t}$ is the raw count of the term t in the document d, N is the total number of documents in the collection, while d_f is the count of documents that contain the term t.

Kullback-Leibler divergence. The Kullback-Leibler divergence (KLD) estimates the similarity between a document and query by measuring the relative entropy between their corresponding models [7,16]. This measure has also been used as a means of identifying terms for which that document is most likely to retrieved [3]. Based on this premise, we use KLD to assign a term a weight that indicates its significant in a document,

$$\text{KLD}(t, d, c) = P(t|d) \log \frac{P(t|d)}{P(t|c)} \tag{1}$$

where $P(t|M)$ which computes the probability of the term t in the model M obtained as $P(t|M) = f_{M,t}/|M|$, and c is the collection-wide model. Büttcher et al. [3] found that Equation 1 favours high-frequency terms in a document and that dampening the probability of a term in the document means that more effective results are obtained.

$$\text{KLD}(t, d, c) = (P(t|d))^{1-\delta} \left(\log \frac{P(t|d)}{P(t|c)}\right)^{1+\delta}$$

Having established means of weighting terms, the weight of a sentence can be computed as the sum of the weights of all the terms it contains,

$$w(s_i) = \sum_{t \in s_i \wedge t \notin stoplist} w(t) \tag{2}$$

where $w(t)$ can be either KLD or TFIDF weight. Excluded from contributing are *stop words*[1].

An obvious shortcoming of the above sentence scoring approach is that it is biased in favor of long sentences (they will have higher score). To account for this bias we normalized sentence weight by sentence length, thus:

$$w_n(s_i) = \frac{w(s_i)}{|s_i|}$$

Sentences are then ordered in descending order of this normalized weight.

Document pruning. Once sentences are in decreasing order of perceived usefulness, the documents must be pruned. Two obvious pruning schemes are *fixed percentage* and *fixed number*, which retain, per document, a user-specified top percentage of sentences or a fixed number of sentences, respectively. We also considered a third method, *weight threshold* where only sentences with weight greater than a predetermined threshold τ are retained. The threshold τ can be established on a per document basis or determined globally for the documents in a collection.

In all the experiments discussed in this paper we made use of the fixed percentage scheme as it was easy to implement and, in preliminary experiments, we found it to give similar results to the fixed number method. We intend to investigate the weight threshold scheme and hybrid methods in future work.

5 Experimental Setup

All experiments were conducted on two collections from the TREC corpus: WT10G, a 10 GB collection of web crawl; and a larger collection, WT100G, a 100 GB crawl of web data both compiled between 1997 and 1999. All queries were taken from two large Excite^{TM} query logs, both of which were compiled at around the same time the collections were crawled. From each query log, 10,000 random unique queries were selected. Queries were only included if they had at least one document retrieved. In addition, we also report results for the WT10G collection using available TREC topics. Here, titles of the Web Track Topics 450-551 are used as queries.

To generate snippets, each document was parsed to remove markup and script tags as these are of no use when generating snippets. Each document was then converted into a strictly alternating sequence of words and punctuations. Whitespace is considered to be punctuation. A word was defined as a set of alphanumeric characters separated by whitespace, or a combination of whitespace

[1] The stop list was under `www.csse.unimelb.edu.au/~jz/resources`

and punctuation. Redundant white space and non-ASCII characters were also removed. All that remained of a document was the text content that would be visible to a user viewing the page with a browser.

Sentences were identified as sequences of terms and punctuation until the punctuation contained an end of sentence marker (one of ?! or .). In addition to these, HTML tags such as `<h1>`, `</h1>`, `<p>`, `</p>`, and `<br/>` were used to assist in identifying the end of sentences. Due to the unstructured nature of web text, a document may contain text with no end of sentence punctuation or HTML tags. Accordingly, we applied a length threshold, restricting sentence length to at most 30 words. Also, if an end of sentence marker appears before the fifth word in a sentence then it is ignored and the words are joined to the following sentence.

To generate a snippet for a query from a given document, a sentence s_i is scored against the query using a scoring function $\sigma(s_i)$. The three sentences with the highest $\sigma(s_i)$, and the title of the document are presented as a summary on the results page. The function $\sigma(s_i)$ we use is based on that used by Turpin et al. [14], but recast as a a simple sorting function. It sorts sentences using the count of unique query terms in the sentence (μ) as the primary key, and the longest span of query terms in a sentence (ς) as the secondary key to break ties. In the sentence scoring scheme proposed by Turpin et al. [14], the total count of query terms in a sentence was used as a tertiary sort key (although they do not describe their method in this fashion). However, in a small study conducted as part of this work, we found that this component tends to favor sentences that contain query terms repeated many times, a large portion of which are spam sentences, and not useful for the purpose of summary generation. Alternately, we use the TFIDF score of a sentence as the tertiary sort key.

To maximize the information held in a snippet, duplicate sentences should be removed. Two sentences are considered duplicate if they have substantial syntactic overlap. More precisely, we slide a window of size n over each document and, for each sentence, count the number of word n-grams that have been previously observed in the same document. If a sentence contains over 80% of previously observed n-grams, it is removed. This process has the effect of removing duplicate and near-duplicate sentences, but it might also remove some useful sentences that coincidently contain 80% of n-grams from the document text proceeding a sentence. In the experiments reported here we use $n = 5$.

Evaluation. Our goal in this paper is to assess whether our proposed pruning schemes generate worse snippets than would have been generated had the full documents been present. Rather than employing human judges to make this decision, as is the case with previous snippet evaluation [13,15], we use simple textual comparison. Snippets generated using the full (unpruned) document are taken as ideal. We generate snippets using the pruned documents and compare them to those generated using the full documents. To evaluate the effectiveness of the pruned surrogates, we count the number of pruned documents which produce snippets *identical* to those generated using the full documents. In addition, we also report the count of *equivalent* snippets, that is, those that have identical

query biased snippet scores. Snippets that are neither identical nor equivalent are called a *miss*.

6 Results

The collections described in Section 5 were indexed using the Zettair search engine[2], and for each query the top 20 documents were retrieved using a Dirichlet-smoothed language model. Removed from a query's result list are duplicate documents, which produce identical snippets and title for a given query. Table 1 is a summary of the statistics of the retrieved documents. Sentences in each document were reordered using the two schemes proposed, TFIDF and KLD, and unordered version of each document were retained. To create surrogates, the leading 60%, 40%, and 20% of the sentences in reordered documents were retained.

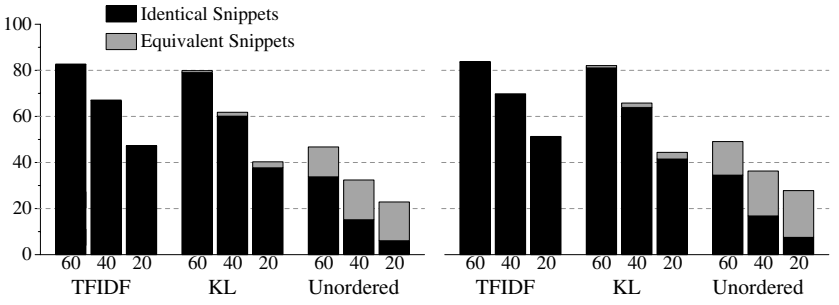
Table 1. Documents retrieved and used for snippet generation. The "Unique" column contains the count of distinct documents used for each collection-query set.

Collection–Queries	Total	Duplicates	Unique	Av. length (KB)
WT10G–TREC451–550	1,948	36	1,790	10.7
WT10G–Excite97	179,456	13,409	96,540	12.3
WT100G–Excite99	180,950	42,219	105,496	14.5

Figure 1 shows the percentage of pruned documents that produced identical (dark bars) and equivalent (light bars) snippets. Compared to pruning without reordering, documents pruned after either TFIDF or KLD reordering produce almost twice as many identical snippets. To further investigate the effect of sentence reordering, we observed the size of ordered and unordered documents processed to find candidate sentences included in the document's snippet. We took the full reordered and unordered documents from WT10G–TREC451-550 (the first row in Table 1), and clustered them into bins according to the number of sentences they contained. Then, for each document, we determined the position of the last sentence that was included in the final snippet. In a bin the average position of the last sentence used was computed. If reordering does indeed cluster sentences useful for snippet generation at the start of the document, then the portion of reordered documents inspected to find the snippet sentences should be smaller than unordered documents. Our findings are illustrated in Figure 2.

On average 70%–80% of sentences must be inspected in each unordered document to generate a snippet and reordered documents produce the same snippets by only traversing the leading 40%–50% sentences in those documents. These findings are consistent in the other two collection-query pairs; however, we omit those results for lack of space.

[2] See `http://www.seg.rmit.edu.au/zettair`



(a) WT10G - Excite97 (b) WT100G - Excite99

Fig. 1. Quality of snippets generated from WT10G for Topics 451-550. Documents were reordered using the various reordering schemes and then pruned. The dark bars indicate the percentage of pruned documents with identical snippets to the full documents while the light colored bar indicates those that generated snippets with the same QBS score.

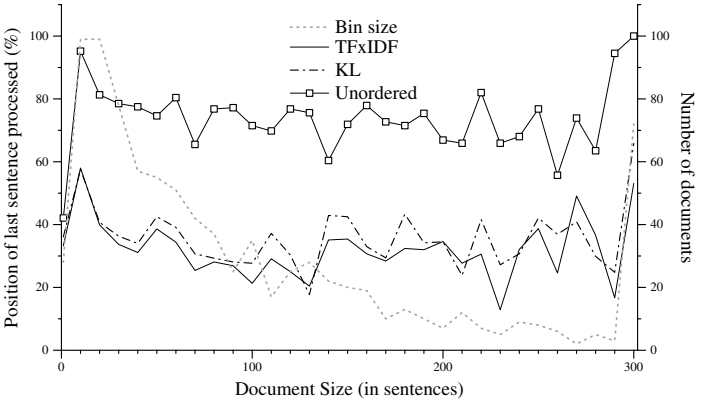


Fig. 2. Average percentage of sentences requiring inspection to locate the ideal sentences for snippet generation. The second ordinate should be used for bin-sizes only.

While the results of Figure 1 are encouraging, the potential degradation in snippet quality due to pruning prompted us to investigate ways for the system to detect when poorer snippets are being produced. In these cases, snippets can be generated from the whole documents stored on disk.

Analysis of miss documents revealed that, on average, 42% of the snippets for these documents included one or more sentences containing no query terms, despite the sentences in the full document snippet all containing one or more query term. Thus, if a snippet contains a sentence with no query term and the total number of the query terms in the snippet is less than the total number of query terms in the index, then we can determine whether sentences containing query terms were removed from the pruned document. In such cases the full document is fetched from disk and used to generate a more accurate snippet. We call this approach *simple go-back* (SGB).

Table 2. Percentage of snippets produced that are identical to those produced with whole documents, and the amount of data (percentage of whole documents) processed per snippet, with and without SGB

Collection–queries	Identical Snipp. (%)		Size (%)	
	No SGB	SGB	No SGB	SGB
WT10G–TOP451-550	69.32	89.39	41.48	62.58
WT10G–Excite97	66.89	82.63	40.78	51.93
WT100G–Excite99	69.49	80.74	40.62	47.71

By incorporating SGB into the snippet generation process, the percentage of documents producing identical snippets is increased on average by 18%. Table 2 shows the increase in the percentage of document producing identical snippets brought by using SGB. However, as full documents are now occasionally fetched, the total amount of data processed also increases, by 13% on average. Note that this is still significantly less than the 80% processed when full, unordered documents are used.

Where it is essential to have snippets that are identical to those that would be produced from the full document, a term-level index, storing term offsets within a document, rather than a document level index with SGB, can be used. Note to identify sentence borders we must also index end of sentence markers. The index can be used to obtain the query-biased score of the candidate snippet sentences prior to inspecting the document. When the pruned document fails to produce snippets with the same query-biased score, the full document should be used.

7 Discussion

Turpin et al. show that by caching on average 3% of the collection size, 98–100% of summarized documents can be retrieved from memory averting the need to go to disk [14]. In this paper we have shown that retaining 40% of the size of documents functions as a good surrogate for majority of documents summarised. We now demonstrate the advantages of caching pruned documents. In particular we compare the benefit of caching pruned compressed documents against caching full compressed documents. To compare our results with Turpin et al.'s work, we conduct our analysis on the WT100G collection. After cleaning – removing markups and using a fixed set of punctuation – the collection size is reduced to 45 GB.

Following from the work of Turpin et al., caching 3% of unpruned documents in a collection would require on average 1.35 GB of random access memory.

To increase the portion of collection stored in cache or to reduce the amount of memory used for document caching, documents can be compressed. Using a scheme with relatively fast decoding speed – such as term-based Huffman coding which has a 50-60% compression ratio [14], the 3% portion can be compressed down to around 0.756 GB. Therefore, over 40% more documents can now be cached by simply applying compression.

In the above schemes, entire documents were being cached. However, as we have shown in this paper, retaining around 40% of a document size functions as good surrogate for a considerable portion of the documents summarised. Moreover, Table 2 shows that to achieve around 80% identical snippets with SGB, on average around 47% of a document would need to be retained. By randomly sampling WT100G collection, where documents were pruned to 48% of their original size, 3% of the collection can be cached in 0.675 GB. This is 11% less RAM than the compression-only scheme.

Of course, pruned documents can also be compressed. Applying similar compression as the previous scheme reduces the cache size required to maintain 3% of the collection to around 0.378 GB: less than a 30% of the size of the original cache and almost half the size of the compressed document cache. By applying a combination of pruning and compression, the entire WT100G collection can now be cached in under 13 GB of memory.

The use of SGB requires 13% more data to be fetched from disk. However, this is well counterbalanced by the increased caching.

8 Conclusion

In this paper we proposed a document pruning strategy, for efficient snippet generation. Our pruning scheme reorders the sentences in each document and retains less than half of the content of each document as surrogates for snippet generation. Furthermore, we describe the Simple Go-Back approach to detect some cases when snippets generated from the pruned surrogates differ form those generated from the full document. Using the system, over 80% of these surrogates produce identical snippets as the full documents.

The implications of the proposed pruning scheme are two-fold. Firstly, compared to using the full documents, the amount of data fetched from disk and processed to generate snippets when using pruned documents is cut by over 50%. Secondly, we have shown that the pruned documents can be cached in less than 30% of the memory required to cache full documents. The additional available memory may be dedicated to caching other components of the query evaluation process, such as results pages or inverted lists.

References

1. Baeza-Yates, R.A., Ribeiro-Neto, B.: Modern Information Retrieval. Addison-Wesley Longman Publishing Co., Inc., Boston (1999)
2. Billerbeck, B., Zobel, J.: Efficient query expansion with auxiliary data structures. Inf. Syst. 31(7), 573–584 (2006)
3. Büttcher, S., Clarke, C.L.A.: A document-centric approach to static index pruning in text retrieval systems. In: CIKM 2006: Proceedings of the 15th ACM international conference on Information and knowledge management, pp. 182–189 (2006)
4. Carpineto, C., de Mori, R., Romano, G., Bigi, B.: An information-theoretic approach to automatic query expansion. ACM Transactions Information Systems 19(1), 1–27 (2001)

5. de Moura, E.S., dos Santos, C.F., Fernandes, D.R., Silva, A.S., Calado, P., Nascimento, M.A.: Improving web search efficiency via a locality based static pruning method. In: WWW 2005: Proceedings of the 14th international conference on World Wide Web, pp. 235–244 (2005)
6. Goldstein, J., Kantrowitz, M., Mittal, V., Carbonell, J.: Summarizing text documents: sentence selection and evaluation metrics. In: ACM SIGIR 1999, pp. 121–128 (1999)
7. Kullback, S., Leibler, R.A.: On information and sufficiency. The Annals of Mathematical Statistics 22(1), 79–86 (1951)
8. Lu, J., Callan, J.: Pruning long documents for distributed information retrieval. In: CIKM 2002: Proceedings of the eleventh international conference on Information and knowledge management, pp. 332–339 (2002)
9. Luhn, H.P.: The automatic creation of literature abstracts. IBM Journal of Research and Development 2(2), 159–165 (1958)
10. Mani, I.: Automatic Summarization. John Benjamins Publishing Company, Amsterdam (2001)
11. Sakai, T., Sparck-Jones, K.: Generic summaries for indexing in information retrieval. In: ACM SIGIR 2001, pp. 190–198 (2001)
12. Silber, H.G., McCoy, K.F.: Efficient text summarization using lexical chains. In: IUI 2000: Proceedings of the 5th international conference on Intelligent user interfaces, pp. 252–255 (2000)
13. Tombros, A., Sanderson, M.: Advantages of query biased summaries in information retrieval. In: ACM SIGIR 1998, pp. 2–10 (1998)
14. Turpin, A., Tsegay, Y., Hawking, D., Williams, H.E.: Fast generation of result snippets in web search. In: ACM SIGIR 2007, pp. 127–134 (2007)
15. White, R.W., Jose, J.M., Ruthven, I.: A task-oriented study on the influencing effects of query-biased summarisation in web searching. Inf. Process. Manage. 39(5), 707–733 (2003)
16. Zhai, C., Lafferty, J.: A study of smoothing methods for language models applied to information retrieval. ACM Trans. Inf. Syst. 22(2), 179–214 (2004)
17. Zobel, J., Moffat, A.: Exploring the similarity space. SIGIR Forum 32(1), 18–34 (1998)
18. Zobel, J., Moffat, A.: Inverted files for text search engines. ACM Comput. Surv. 38(2), 6 (2006)