Ranked Document Retrieval in (Almost) No Space [⋆]

Nieves R. Brisaboa[1], Ana Cerdeira-Pena[1], Gonzalo Navarro[2] and Óscar Pedreira[1]

[1] Database Lab., Univ. of A Coruña, Spain.
{brisaboa,acerdeira,opedreira}@udc.es
[2] Dept. of Computer Science, Univ. of Chile. gnavarro@dcc.uchile.cl

Abstract. Ranked document retrieval is a fundamental task in search engines. Such queries are solved with inverted indexes that require additional 45%-80% of the compressed text space, and take tens to hundreds of microseconds per query. In this paper we show how ranked document retrieval queries can be solved within tens of milliseconds using essentially *no extra space* over an in-memory compressed representation of the document collection. More precisely, we enhance wavelet trees on bytecodes (WTBCs), a data structure that rearranges the bytes of the compressed collection, so that they support ranked conjunctive and disjunctive queries, using just 6%–18% of the compressed text space.

1 Introduction

Ranked document retrieval on a large collection of natural-language text documents is the key task of a search engine. Given a query composed of a set of terms, the engine returns a list of the documents most relevant to the query.

Efficient ranked document retrieval relies on the use of inverted indexes [1–3]. Given a query, the system computes the union (*bag-of-words*) or intersection (*weighted conjunctive*) of the posting lists of the terms composing the query, keeping only the documents with highest relevance with respect to the query.

The inverted index does not support by itself all the operations needed in a search engine. For example, showing snippets or cached versions of results. This requires storing the text of the documents. Compressing the text and the inverted index is useful not only to save space, but it also reduces the amount of I/O needed to answer queries on disk-based systems. A recent trend (e.g., [3–6]) is to maintain all the data in main memory, of a single machine or a cluster.

The texts of the documents are usually stored in a compressed form that allows fast decompression of random portions of the text. Such compressors achieve 25%–30% of the size of the original text. Inverted indexes are also compressed, and amount to an additional 15%–20% of the size of the original text,

[⋆] Partially funded by MICINN (PGE and FEDER) grant TIN2009-14560-C03-02, and by Xunta de Galicia (co-funded with FEDER) ref. 2010/17, for the Spanish group; by MICINN FPU program, ref. AP2007-02484 for the second author; and by Fondecyt grant 1-110066, Chile, for the third author.

or 45%–80% of the size of the compressed text [1, 2, 7, 8]. Typical query times of in-memory systems are in the orders of tens to hundreds of microseconds.

A recent alternative to storing the text plus the inverted index is the *Wavelet Tree on Bytecodes (WTBC)* [9]. Within the space of the compressed text (i.e., around 30%–34% of the text size) the WTBC not only can extract arbitrary snippets or documents, but it also solves *full-text* queries. Full-text queries are usually solved with a *positional* inverted index, which stores exact word positions, yet this is outperformed by the WTBC when little space over the compressed text is available. The representation was later extended to *document retrieval* queries, that is, listing all the distinct documents where a query appears [10]. However, *ranked document retrieval* queries, which are arguably the most important ones for the end-user, have not been addressed under this scheme.

In this paper we close this gap. We show how WTBCs can be extended to efficiently support ranked document retrieval queries as well. As a result, all the main IR queries can be carried out on top of a data structure that requires just 6%-18% on top of the compressed text space (2.0%–5.5% of the original text space). The times of the WTBC to solve ranked document retrieval queries are in the order of milliseconds, which is significantly higher than inverted index times. However, those times are still reasonable in many scenarios and the solution offers important space advantages compared to the 45%–80% of extra space posed by inverted indexes, which may be key to avoid using secondary storage, to use fewer machines, or even to achieve a feasible solution when the memory is limited (as in mobile devices).

2 WTBC: Wavelet Trees on Bytecodes

The *Wavelet Tree on Bytecodes* (WTBC) [9] is a method for representing natural language texts in a compressed form, so that random access to any portion of the text, and search for the occurrences of any term, are efficiently supported. The WTBC is built on a text compressed using any word-based byte-oriented semistatic statistical compressor, by rearranging the codewords into a wavelet-tree-like structure.

The basic idea in the WTBC is to rearrange the text by placing the different bytes of each codeword in different nodes of a tree. The root of the tree is an array containing the first byte of the codeword of each word in the text, in the same order they appear in the original text. The second byte of each codeword (having more than one byte) is placed on the second level of the tree, and so on.

The main operations in a WTBC are *decoding* the word at a given position, *locating* the occurrences, and *counting* the number of occurrences of a word. These algorithms are based on the use of *rank* and *select* operations over the bytemap of each node of the tree. Partial counters are maintained for each bytemap in order to efficiently compute *rank* and *select* while posing just a 3% space overhead over the original text size [9].

3 Efficient Ranked Document Retrieval

In this section we present our proposal for solving ranked document retrieval queries using the WTBC over (s,c)-DC [11]. We concatenate all documents of the corpus in a single text string. We assume that each document ends with a special symbol `'$'`, which then becomes a document separator (just as in [10]). Then, the string is compressed with (s,c)-DC and a WTBC is built on the result of the compression. For efficiency reasons, we reserve the first codeword of the (s,c)-DC encoding scheme for the `'$'` symbol, so the document separator can be easily found in the root of the tree, since its codeword has only one byte.

We consider top-k *conjunctive* and *bag-of-words* queries. We have developed two different alternatives, namely WTBC-DR and WTBC-DRB, depending on whether or not we use additional space over the WTBC. Due to space constraints, this paper focuses on WTBC-DR, but a complete study presenting both approaches can be accessed through `http://arxiv.org/abs/1207.5425`.

We use the *tf-idf* relevance measure. The document frequency values are stored, one per word, in our index within insignificant extra space, as the vocabulary size becomes irrelevant as the collection grows [12].

3.1 Solution with No Extra Space (WTBC-DR)

The procedure uses a priority queue storing *segments*, that is, concatenations of consecutive documents. The priority will be given by the *tf-idf* value of the concatenations (seen as a single document). We start by inserting in the queue the segment that corresponds to the concatenation of all the documents, with its associated priority obtained by computing its *tf-idf* value. A segment is represented by the corresponding endpoints in the root bytemap, $T[1,n]$, of the WTBC. Since the *idf* of each word is precomputed, to compute *tf-idf* relevance value we only need to calculate the *tf* of each word in the segment, that is, we *count* its number of occurrences in the segment.

The procedure repeatedly extracts the head of the queue (the first time, we extract the segment $T[1,n]$). If the extracted segment has more than one document, the procedure splits it into two subsegments, by using the `'$'` symbol closest to the middle of the segment, as the point to divide it. This `'$'` is easily found using *rank* and *select* on T (i.e., for a segment $T[a,b]$, we use, roughly, $select_\$(T, rank_\$(T,(a+b)/2)))$, which also tell us the number of documents in each subsegment. After the division, the relevance of each of the two subsegments is computed, and they are inserted in the queue using their relevance as priority.

If, instead, the extracted segment contains only one document, it is directly output (with its *tf-idf* relevance value), as the next most relevant document. This is correct because *tf-idf* is monotonic over the concatenation: the *tf-idf* of the concatenation of two documents is not smaller than the *tf-idf* of any of them. Thus the relevance of the individual document extracted is not lower than that of any other remaining in the priority queue.

The iterative process continues until we have output k documents. In this way, it is not necessary to process all the documents in the collection, but rather

Algorithm 1: ranked bag-of-words retrieval with WTBC-DR

```
Input: $wt$ (WTBC), $q$ (query), $k$ (top-$k$ limit)
Output: list of top-$k$ ranked documents
$s.start\_pos \leftarrow 1$; $s.end\_pos \leftarrow n$; $s.score \leftarrow tfidf(s,q)$; $s.ndocs \leftarrow N$;
$pq \leftarrow \langle\rangle$; $insert(pq,s)$ // $s.score$ is the priority for queue $pq$ ;
while less than $k$ documents output and $\neg empty(pq)$ do
    $s \leftarrow pull(pq)$;
    if $s.ndocs = 1$ then
        output $s$
    else
        $\langle s_1, s_2\rangle \leftarrow split(s)$ // computes $s_i.start\_pos$, $s_i.end\_pos$ and $s_i.ndocs$;
        $s_1.score \leftarrow tfidf(s_1,q)$; $s_2.score \leftarrow s.score - s_1.score$;
        $insert(pq,s_1)$; $insert(pq,s_2)$;
    end
end
```

the search is guided towards the areas that contain the most promising documents for the query until it finds the top-k answers. Note that the procedure does not need to know k beforehand; it can be stopped at any time. The pseudocode for bag-of-words queries is given in Algorithm 1. For weighted conjunctive queries we add an additional check during the procedure: if a segment does not contain some of the words in the query (i.e., some *tf* is zero), the segment is discarded without further processing.

4 Experimental Evaluation

We evaluated the performance of the proposed WTBC-DR algorithms over a data set created by aggregating text collections from TREC-2: AP Newswire 1988, and Ziff Data 1989-1990, and TREC-4, namely Congressional Record 1993, and Financial Times 1991 to 1994. All of them form a document corpus of approximately 1GB (ALL).

Table 1 (left) gives the main statistics of the collection used, as well as the results obtained when it is represented with WTBC over (s,c)-DC, for the WTBC-DR variant (right). We show the compression ratio (CR) (in % of the size of the original text collection), together with the time to create the structures (CT) and to recover the whole text back from them (DT), in seconds. The raw compressed data uses around 32.5% of the space used by the plain text, and the WTBC just requires an additional waste of 2.5% of extra space for the bytemap *rank* and *select* operations, for a total of 35%. Within that amount of space WTBC-DR is able to perform ranked document retrieval[1].

[1] An additional 3% of space would be needed in case of WTBC-DRB alternative. See `http://arxiv.org/abs/1207.5425` for more details.

Table 1. Description of the corpus used and compression properties

Corpus	size (MB)	#docs	#words	voc. size
ALL	987.43	345,778	219,255,137	718,691

Index	CR	CT	DT
WTBC-DR	35.0	40.1	8.6

4.1 Ranked Document Retrieval

Table 2 shows the average times[2] (in milliseconds) to find the top-k (using $k = 10$ and $k = 20$) ranked documents for a set of queries, using WTBC-DR[3]. We considered different sets of queries. First, we generated synthetic sets of queries, depending on the document frequency of the words (f_{doc}): *i)* $10 \leq f_{doc} \leq 100$, *ii)* $101 \leq f_{doc} \leq 1{,}000$, *iii)* $1{,}001 \leq f_{doc} \leq 10{,}000$, and *iv)* $10{,}001 \leq f_{doc} \leq 100{,}000$, and also on the number of words that compose a query, namely, 1, 2, 3, 4 and 6. Each set is composed of 200 queries of words randomly chosen from the vocabulary of the corpus, among those belonging to a specific range of document frequency. Second, we also used queries from a real query log[4] (*real*), and created 5 sets of 200 queries randomly chosen composed of 1, 2, 3, 4, and 6 words, respectively. The same sets of queries were used for dealing with both conjunctive and bag-of-words scenarios.

In general, we can observe that, with essentially *no extra space*, all queries are solved by WTBC-DR within tens of milliseconds. More in detail, in conjunctive queries, the processing times decrease as the number of words in the query increases, within a given f_{doc} band. This is expected when the words are chosen independently at random, since more words give more pruning opportunities. However, in the scenario *iv*), where words appear in too many documents, and in *real*, where the query words are not independent, the WTBC-DR pruning is not efficient enough and its times grow with the number of words. On the other hand, if we consider the bag-of-words scenario, the more query words, the higher is the average processing time, since each word increases the number of valid documents. The same behavior applies for *real* queries, independently of the document frequency of the words composing the query.

5 Conclusions

We have shown how the WTBC, a compressed data structure that supports full-text searching and document retrieval within essentially the space of the compressed text, can be enhanced to support also ranked document retrieval, which is by far the most important operation in IR systems, requiring just tens of milliseconds. The enhanced WTBC becomes a very appealing solution in scenarios where minimizing the use of main memory is of interest, as it supports all the typical repertoire of IR operations at basically no storage cost.

[2] We used an AMD Phenom II X4 955 Processor (3.2 GHz) and 8GB RAM.

[3] We refer the reader to `http://arxiv.org/abs/1207.5425` for a complete analysis comparing both WTBC-DR and WTBC-DRB performance.

[4] Obtained from TREC (`http://trec.nist.gov/data/million.query.html`)

Table 2. Results for top-10 and top-20 1-word queries, conjunctive queries (AND) and bag-of-words queries (OR)

$\mathtt{f}_{doc}$	Query	#words per query									
		1		2		3		4		6	
	type	top-10	top-20	top-10	top-20	top-10	top-20	top-10	top-20	top-10	top-20
i)	AND	2.27	3.45	0.86	0.87	0.55	0.55	0.43	0.42	0.28	0.28
	OR			3.86	4.91	4.28	5.91	5.17	7.35	6.86	9.53
ii)	AND	6.18	7.80	9.57	9.61	6.54	6.55	4.70	4.71	3.44	3.44
	OR			10.12	13.74	14.22	19.74	18.56	24.53	27.78	35.91
iii)	AND	15.06	18.62	63.05	72.37	64.06	66.67	53.51	53.63	44.19	46.31
	OR			29.63	38.54	43.20	57.81	61.65	78.06	94.96	118.71
iv)	AND	53.40	66.15	151.16	185.76	284.92	341.42	382.92	415.76	404.26	410.35
	OR			98.84	125.52	156.79	202.71	223.31	281.07	359.84	462.16
real	AND	6.68	9.08	34.92	41.55	67.36	77.52	78.34	87.11	101.22	108.95
	OR			27.96	36.17	58.91	75.74	82.80	106.42	150.94	192.02

This paper presents one of the two proposals we have developed. In particular, the one that does not use any extra space on top of the WTBC. This proposal, called WTBC-DR, applies a prioritized traversal by relevance, and solves bag-of-words (disjunctive) queries and weighted conjunctive queries within milliseconds.

References

1. Zobel, J., Moffat, A.: Inverted files for text search engines. ACM Comp. Surv. **38**(2) (2006)
2. Baeza-Yates, R., Ribeiro-Neto, B.: Modern Information Retrieval. 2nd edn. Addison-Wesley (2011)
3. B. Croft, D.M., T.Strohman: Search Engines: Information Retrieval in Practice. Pearson Education (2009)
4. Strohman, T., Croft, B.: Efficient document retrieval in main memory. In: Proc. 30th SIGIR. (2007) 175–182
5. Transier, F., Sanders, P.: Engineering basic algorithms of an in-memory text search engine. ACM Trans. Inf. Sys. **29**(1) (2010) 2:1–2:37
6. Culpepper, S., Moffat, A.: Efficient set intersection for inverted indexing. ACM Trans. Inf. Sys. **29**(1) (2010)
7. Witten, I., Moffat, A., Bell, T.: Managing Gigabytes. 2^{nd} edn. Morgan Kaufmann Publishers (1999)
8. Baeza-Yates, R., Moffat, A., Navarro, G.: Searching large text collections. In: Handbook of Massive Data Sets, Kluwer Academic Publishers (2002) 195–244
9. Brisaboa, N., Fariña, A., Ladra, S., Navarro, G.: Implicit indexing of natural language text by reorganizing bytecodes. Inf. Retr. (2012) Av. online.
10. Arroyuelo, D., González, S., Oyarzún, M.: Compressed self-indices supporting conjunctive queries on document collections. In: Proc. 17th SPIRE. (2010) 43–54
11. Brisaboa, N., Fariña, A., Navarro, G., Paramá, J.: Lightweight natural language text compression. Inf. Retr. **10**(1) (2007) 1–33
12. Heaps, H.: Information Retrieval - Computational and Theoretical Aspects. Academic Press (1978)