Information Processing and Management xxx (xxxx) xxx–xxx

Contents lists available at ScienceDirect

Information Processing and Management

journal homepage: www.elsevier.com/locate/infoproman

Hybrid compression of inverted lists for reordered document collections[☆]

Diego Arroyuelo[*,1,a], Mauricio Oyarzún[b], Senén González[c], Victor Sepulveda[d]

[a] *Department of Informatics, Universidad Técnica Federico Santa María, Chile*
[b] *Universidad Arturo Prat – Iquique, Chile*
[c] *Software Competence Center Hagenberg GmbH, Austria*
[d] *Department of Computer Science, University of Chile*

ARTICLE INFO

Keywords:
Index compression for information retrieval
Reordered document collections

ABSTRACT

Text search engines are a fundamental tool nowadays. Their efficiency relies on a popular and simple data structure: *inverted indexes*. They store an *inverted list* per term of the vocabulary. The inverted list of a given term stores, among other things, the document identifiers (docIDs) of the documents that contain the term. Currently, inverted indexes can be stored efficiently using integer compression schemes. Previous research also studied how an optimized document ordering can be used to assign docIDs to the document database. This yields important improvements in index compression and query processing time. In this paper we show that using a hybrid compression approach on the inverted lists is more effective in this scenario, with two main contributions:

- First, we introduce a document reordering approach that aims at generating runs of consecutive docIDs in a properly-selected subset of inverted lists of the index.
- Second, we introduce hybrid compression approaches that combine gap and run-length encodings within inverted lists, in order to take advantage not only from small gaps, but also from long runs of consecutive docIDs generated by our document reordering approach.

Our experimental results indicate a reduction of about 10%–30% in the space usage of the whole index (just regarding docIDs), compared with the most efficient state-of-the-art results. Also, decompression speed is up to 1.22 times faster if the runs of consecutive docIDs must be explicitly decompressed, and up to 4.58 times faster if implicit decompression of these runs is allowed (e.g., representing the runs as intervals in the output). Finally, we also improve the query processing time of AND queries (by up to 12%), WAND queries (by up to 23%), and full (non-ranked) OR queries (by up to 86%), outperforming the best existing approaches.

1. Introduction

Inverted indexes are the *de facto* data structure to support the high-efficiency requirements of a text search engine (Baeza-Yates & Ribeiro-Neto, 2011; Büttcher, Clarke, & Cormack, 2010; Manning, Raghavan, & Schütze, 2008; Witten, Moffat, & Bell, 1999; Zobel &

[☆] A preliminary version of this paper appeared in Proc. of the 36th International ACM SIGIR conference on research and development in Information Retrieval (SIGIR'13).
[*] Corresponding author.
E-mail addresses: darroyue@inf.utfsm.cl (D. Arroyuelo), moyarzunsil@unap.cl (M. Oyarzún), sgonzale@dcc.uchile.cl (S. González), vsepulve@dcc.uchile.cl (V. Sepulveda).
[1] Funded in part by FONDECYT Grant 11121556, Chile.

https://doi.org/10.1016/j.ipm.2018.05.007
Received 17 October 2017; Received in revised form 31 March 2018; Accepted 18 May 2018
0306-4573/ © 2018 Elsevier Ltd. All rights reserved.

Moffat, 2006). This includes, for instance, providing fast response to thousands of queries per second, while using as less server memory as possible, among others (Dean, 2009). Given a document collection $\mathscr{D}$ with vocabulary $\Sigma = \{w_1, \cdots, w_V\}$ of V different words (or terms), an inverted index for $\mathscr{D}$ stores a set of *inverted lists* $I_{w_1}[1..n_1], \cdots, I_{w_V}[1..n_V]$. Every list $I_{w_i}[1..n_i]$ stores a *posting* for each of the n_i documents that contain the term $w_i \in \Sigma$. Typically, a posting stores the document identifier (docID) of the document that contains the term, the number of occurrences of the term in this document (the term frequency) and, in some cases, the positions of the occurrences of the term within the document. The inverted index also stores a *vocabulary table*, which allows us to access the respective inverted lists.

1.1. *Inverted index compression*

Inverted lists tend to be big, occupying important amounts of memory space. Also, transferring them from secondary storage can take considerable time, degrading query processing time. Hence, lists are compressed, not only to reduce their space usage, but also to reduce the transference time from disk—which can be up to 4–8 times slower if the disk-resident lists are not compressed (Büttcher et al., 2010, see Table 6.9–page 213). To answer a query, the involved lists (or parts of them) must be decompressed. Therefore, fast decompression is a key issue to support quick answers.

Inverted index compression has been studied in depth in the literature (Baeza-Yates & Ribeiro-Neto, 2011; Büttcher et al., 2010; Manning et al., 2008; Witten et al., 1999). Usually, docIDs, frequencies, and positions are stored separately, using different inverted-list layers. Thus, each layer can be compressed independently, using the best compression scheme for each case. Even though it is important to compress each of these components, this paper is exclusively devoted to compress docIDs. Frequencies and term positions can be usually compressed using similar techniques. According to Yan, Ding, and Suel (2009) and Arroyuelo, González, Marin, Oyarzún, and Suel (2012), the space used by docIDs correspond to about 65% of a docIDs + frequencies index. If we also consider positional information, docIDs correspond to about 20% of the overall space (Arroyuelo et al., 2012). Besides the space savings, in this paper we are also interested in reducing query processing time.

The most used approach to compress the docIDs of an inverted list I_{w_i} is gap encoding: we first sort the lists by increasing docID, and then represent the list using the differences between consecutive docIDs (minus 1, for technical reasons that will be made clear through the paper). We call DGap these differences. Gap encoding usually generates a distribution with small numbers, in particular for long lists. As we shall see through this paper, many of these DGaps are actually 0s, which correspond to terms that appear in documents whose docIDs are consecutive (recall that we subtract 1 to the difference). We call *runs* the list regions containing consecutive docIDs.

To support efficient searches, compressed inverted lists are logically divided into blocks of, say, 128 DGaps. This allows us skipping blocks at search time, decompressing only the blocks that are relevant for a query. Among the existing compression schemes for inverted lists, we have classical encodings like Elias δ and γ (Elias, 1975) and Golomb/Rice (Golomb, 1966), as well as the more recent ones VByte (Williams & Zobel, 1999), Simple 9 (Anh & Moffat, 2005), and PForDelta (Zukowski, Héman, Nes, & Boncz, 2006) encodings. All these methods benefit from sequences of small integers.

1.2. *Document reordering*

The assignment of docIDs to a given document database is not trivial, being an important problem in information retrieval (Barla Cambazoglu & Baeza-Yates, 2015; Büttcher et al., 2010), and databases (Johnson, Krishnan, Chhugani, Kumar, & Venkatasubramanian, 2004; Lemire, Kaser, & Aouiche, 2010). This task—usually known as *document reordering*—sorts the documents in $\mathscr{D}$, to then assign the docIDs following this order. For instance, sorting the documents according to their URLs is a simple and effective method (Silvestri, 2007). These are called *ordered document collections* and will be the focus of this paper. Document reordering is not always feasible, as discussed in Yan et al. (2009). However, there are still many applications where this can be used.

The advantage of assigning docIDs in an optimized way is that it yields smaller DGaps in the inverted lists, hence better compression can be achieved. Although the original problem can be formulated as an instance of the TSP problem (an hence, it is NP-Complete) (Shieh, Chen, Shann, & Chung, 2003), there are several heuristic approaches that achieve a better compression performance (Blanco & Barreiro, 2005; Blandford & Blelloch, 2002; Shieh et al., 2003; Silvestri, 2007; Silvestri, Orlando, & Perego, 2004), and in general a much better inverted index performance (Tonellotto, Macdonald, & Ounis, 2011; Yan et al., 2009). Improvements of up to 50% both in space usage and *Document at a Time* (DAAT from now on) query processing have been reported (Ottaviano & Venturini, 2014; Tonellotto et al., 2011; Yan et al., 2009). This problem has been also studied for databases and bitmap indexes (Johnson et al., 2004; Lemire et al., 2010), and for the particular case of e-Commerce (Ramaswamy, Konow, Trotman, Degenhardt, & Whyte, 2017), among others, with similar conclusions.

A remarkable feature of ordered document collections is that the number of consecutive docIDs within inverted lists is increased, which explains the improvement in space usage. For instance, after reordering the well-known TREC GOV2 document collection, about 60% of the postings correspond to consecutive docIDs, whereas a random ordering yields just 11% of consecutive docIDs.

But, more importantly, these consecutive docIDs actually form long *runs* in the inverted lists. These runs of consecutive docIDs are runs of 0s when gap encoded. Rather than regarding these 0s separately in the encoding, *we propose to deal with the runs as a whole*. Having runs of equal symbols in a sequence allows us to use, for instance, run-length encoding (Golomb, 1966): we simply encode a run writing its length. For inverted lists, however, using run-length encoding would not be as effective as for other applications, as there are only runs of 0s: other values rarely form long runs. To tackle this issue, we introduce compression approaches that mix gap encoding along with run-length encoding. The idea is that run-length encoding is used for the segments where runs occur, leaving gap