On Inverted Index Compression for Search Engine Efficiency

Matteo Catena[1], Craig Macdonald[2], and Iadh Ounis[2]

[1] GSSI - Gran Sasso Science Institute, INFN
Viale F. Crispi 7, 67100 L'Aquila, Italy
`matteo.catena@gssi.infn.it`
[2] School of Computing Science, University of Glasgow,
Glasgow G12 8QQ, UK
`{craig.macdonald,iadh.ounis}@glasgow.ac.uk`

Abstract. Efficient access to the inverted index data structure is a key aspect for a search engine to achieve fast response times to users' queries. While the performance of an information retrieval (IR) system can be enhanced through the compression of its posting lists, there is little recent work in the literature that thoroughly compares and analyses the performance of modern integer compression schemes across different types of posting information (document ids, frequencies, positions). In this paper, we experiment with different modern integer compression algorithms, integrating these into a modern IR system. Through comprehensive experiments conducted on two large, widely used document corpora and large query sets, our results show the benefit of compression for different types of posting information to the space- and time-efficiency of the search engine. Overall, we find that the simple Frame of Reference compression scheme results in the best query response times for all types of posting information. Moreover, we observe that the frequency and position posting information in Web corpora that have large volumes of anchor text are more challenging to compress, yet compression is beneficial in reducing average query response times.

1 Introduction

The ubiquitous *inverted index* data structure remains a key component of modern search engines [1]. Indeed, for each unique indexed term, the inverted index contains a *posting list*, where each *posting* contains the occurrences information (e.g. frequencies, and positions) for documents that contain the term. To rank the documents in response to a query, the posting lists for the terms of the query must be traversed, which can be costly, especially for long posting lists.

Different orderings of the postings in the posting lists change both the algorithm for ranked retrieval and the underlying representation of postings in the inverted index, such as how they are compressed. For instance, the postings for each term can be sorted in order of impact, allowing ranked retrieval to be short-circuited once enough documents have been retrieved [2]. However, search engines reportedly [1,3,4] use the traditional static *docid* ordering, where each posting list is ordered by ascending document id, which permits a reduced inverted index size and efficient retrieval [5]. The appropriate compression of this classical docid ordering of posting lists is the focus of our work.

Various schemes have been proposed to ensure the time- and space-efficient compression of the inverted index posting lists. Indeed, as ranking is data-intensive due to the traversing of posting lists, maximising decompression speed while minimising the size of the posting lists is necessary to ensure quick query response times. As docid ordered posting lists contain integer numbers, such compression schemes (or *codecs*) can be operated on a stream of postings – known as *oblivious* – (for example, Elias-Gamma [6]), or on groups of postings – known as *list-adaptive* [7] – (e.g. PForDelta [8]). There are a few comparisons of such integer compression schemes on inverted indexes. Yan et al. [9] analysed the compression ratio and decompression speed of many schemes while dealing with the compression of doc ids and term frequencies extracted from the GOV2 corpus using the AOL query log dataset. A similar work has been done by Lemire et al. [10], studying the codecs behaviour for the ClueWeb09 corpus, but focusing just on document ids. While dealing with smaller datasets, Delbru et al. [11] provided a comparison of codecs integrating these into an actual search engine with positional information. However, missing from the literature is a study that both deploys compression codecs into a realistic information retrieval (IR) system and measures performances on different, large and well-known document corpora and query sets. Moreover, the benefit of different compression algorithms for each of the different types of *payload* data within a posting – ids, term and field frequencies, positions – has not been comprehensively studied.

For the above reasons, this paper provides a thorough comparison and analysis of the response time and index size benefits of modern posting list compression codecs, deployed within an actual modern search engine platform – namely Terrier [12][1] – and examining different posting payload information. Indeed, the contributions of this paper are three-fold: Firstly, we separately analyse the compression benefits of document ids, term frequencies, field frequencies and positions, in terms of average query response time and index size; Secondly, we analyse how different term distributions caused by anchor text affect the achievable compression; Finally, we use these thorough experiments on two standard corpora, namely GOV2 and ClueWeb09 to derive best practices for index compression. The results of our study demonstrate that the contribution of compression in time- and space-efficiency varies greatly depending on the type of posting payload information, while leading us to suggest the use of the simpler Frame of reference (FOR) [24] list-adaptive algorithms for the best benefit to average query response times.

This paper is structured as follows: Section 2 provides a background on efficient ranked retrieval; Section 3 summarises the compression codecs we analyse later within an IR system; Section 4 describes our experimental setup. Sections 5 through 8 show our experimental results; Conclusions and final remarks follow in Section 9.

2 Background

While the effectiveness of a search engine at satisfying the users' information needs is critical, efficiency is also a key aspect for IR, not least because techniques that typically enhance effectiveness can degrade the efficiency of the search engine, i.e. its ability to respond to user queries in a timely fashion [13]. Indeed, efficiency is also important to a search engine, as users are not willing to wait long for queries to be answered [14].

[1] `http://terrier.org`

The architecture of a modern IR system can be roughly divided into three layers [18]: the index with its corresponding *(de)compression* layer; the *matching* layer, which identifies documents to rank from the inverted index; and the uppermost *re-ranking* layer, which applies features and learning to rank techniques to obtain the final ranking of documents.

Techniques to enhance the efficiency of an IR system have been proposed at each layer. For instance, at the matching layer, dynamic pruning techniques such as WAND [15] enhance efficiency by omitting the scoring of documents that cannot reach the final retrieved set. In the top-most re-ranking layer, Cambazoglu et al. [16] showed how learning to rank models could be simplified to enhance their efficiency. Similarly, Wang et al. showed how various features could be avoided [13] or their efficiency cost reduced [17] while minimising any negative impact on effectiveness.

On the other hand, in this paper, we focus on the lowest architecture layer, and in particular, on the compression of the posting lists within the inverted index. Each posting contains a *payload* set of integers, representing the occurrence information of a term within a given document. Indeed, a typical posting contains the docid of the matching document, the frequency of occurrence in the document (often within different *fields*, such as title, URL, body, and anchor text [18]), and – to facilitate phrasal and proximity matching – *positional* information (as a set of ascending integers for *each* posting).

The compression of the posting lists within the inverted index ensures that as much as possible of the inverted index can be kept in the higher levels of the computer memory hierarchy – indeed, many search engines reportedly keep the entire inverted index in main memory [1], as in our experiments. Hence, a compression scheme should not only be time-efficient (i.e. inexpensive to decompress), but also space-efficient (i.e. high compression), to minimise the necessary required computing resources while answering queries. While a range of existing compression schemes or *codecs* have been proposed in the literature (see review in the next section), a comprehensive study addressing their actual impact upon search engine retrieval efficiency has not previously been addressed in the literature. Indeed, while several compression schemes were compared in [9] for document ids and term frequencies extrapolated from the GOV2 corpus against the AOL query log dataset, their study was primarily focused on the decompression speed in terms of integers per second, instead of the resulting impact on query response time. Moreover, they did not consider other posting payload information, such as field frequencies and positions, nor did they investigate how codecs behave on different corpora. On the other hand, while Lemire et al. [10] did investigate compression codecs on both the GOV2 and ClueWeb09 corpora, they were focused only on docid compression, and did not analyse the impact on search engine efficiency in terms of query response time. Finally, while Delbru et al. [11] integrated different compression codecs into an actual IR system with document ids, term frequencies and positional information, their experiments with synthetically generated queries do not necessarily reflect a realistic search engine workload. Hence, in this work, we review a wide-range of posting list compression codecs from the recent literature (Section 3) and later thoroughly empirically compare their time- and space-efficiency for a realistic query workload, across different posting payload information (Sections 5-8).

Table 1. The presented codecs divided by features

Codec	Bitwise	Byte aligned	Word aligned	Oblivious	List-adaptive
Unary/Gamma	✓			✓	
Golomb/Rice	✓				✓
Variable-byte		✓		✓	
Simple family			✓		✓
FOR/PFOR		✓			✓

3 Compression Techniques

Inverted index compression has been common for some time. For example, one common practice while storing a posting list is to use *delta-gaps* (or *d-gaps*) where possible [5], i.e. to record the differences between monotonically increasing components (such as ids or positions) instead of their actual values. Indeed, by using notations different from binary word-aligned, smaller numbers lead to smaller representations.

Many different compression algorithms or *codecs* have been designed to encode integers. We can distinguish codecs using attributes such as the nature of their output, in terms of bitwise, byte-aligned or word-aligned. Moreover, we differentiate between codecs that compress every value on its own, and those which compress values in groups, called *oblivious* and *list-adaptive* methods respectively [7]. In the following, we provide a short description of the oblivious (Section 3.1) and list-adaptive (Section 3.2) codecs that we analyse in this paper. Their attributes are summarised in Table 1.

3.1 Oblivious Codecs

When a set of integers is given to an oblivious codec for compression, it encodes each value on its own, without considering its value relative to the rest of the set. A desirable side effect is that every single value can be decompressed separately, or by the decompression of just the preceding values if d-gaps are used. On the other hand, such codecs ignore global information about the set, which can help to attain a better compression ratio. The oblivious compression algorithms we deploy in our experiments are briefly described below.

Unary and Gamma: Unary and Gamma codecs are two bitwise, oblivious codecs. Unary represents an integer x as $x-1$ one bits and a zero bit (ex.: 4 is 1110). While this can lead to extremely large representations, it is still advantageous to code small values. Gamma, described in [6], represents x as the Unary representation of $1+\lfloor \log_2 x \rfloor$ followed by the binary representation of $x-2^{\lfloor \log_2 x \rfloor}$ (ex.: 9 is 1110 001).

Variable Byte: Variable byte codec [19] is a byte-aligned, oblivious codec. It uses the 7 lower bits of any byte to store a partial binary representation of the integer x. It then marks the highest bit as 0 if another byte is needed to complete the representation, or as 1 if the representation is complete. For example, 201 is 10000001 01001001. While this codec may lead to larger representations, it is usually faster than Gamma in term of decompression speed [20].

3.2 List-Adaptive Codecs

A list-adaptive codec compresses integers in blocks, exploiting aspects such as the proximity of values in the compressed set. This information can be used to improve compression ratio and/or decompression speed. However, this also means that an entire block must be decompressed even when just a single posting is required from it (e.g. for partial scoring approaches such as WAND [15]). Moreover, it is possible to obtain a larger output than the input when there are not enough integers to compress, because extra space is required in the output to store information needed at decompression time. When there are too few integers to be compressed, this header information can be larger in size than the actual payload being compressed. Below, we provide short descriptions of the list-adaptive codecs that we investigate in this work.

Golomb and Rice: Golomb codec is a bitwise and list-adaptive compression scheme [21]. Here, an integer x is divided by b. Then, Unary codec is used to store the quotient q while the remainder r is stored in binary form. b is chosen depending on the integers being compressed. Usually, $b = 0.69 * avg$ where avg is the average value of the numbers being treated [5]. In the Rice codec [22], b is a power of two, which means that bitwise operators can be exploited, permitting more efficient implementations at the cost of a small increase in the size of the compressed data. Nevertheless, Golomb and Rice coding are well-known for their decompression inefficiency [7, 10, 23], and hence, we omit experiments using these codecs from our work.

Simple Family: This family of codecs, firstly described in [23], stores as many integers as possible in a single word. It is possible using the first 4 bits of a word to describe the organisation of the remaining 28 bits. For example, in a word we can store $\{509, 510, 511\}$ as three 9-bits values, with the highest 4 bits of the word reflecting this configuration, at a cost of one wasted bit.

Frame Of Reference (FOR): Proposed by Goldstein et al. [24], FOR compresses integers in blocks of fixed size (e.g. 128 elements). It computes the gap between the maximum M and the minimum m value in the block, then stores m in binary notation. The other elements are saved as the difference between them and m using b bits each, where $b = \lceil \log_2(M + 1 - m) \rceil$.

Patched Frame Of Reference (PFOR): FOR may lead to a poor compression in presence of outliers: single large values that force an increase of the bit width b on all the other elements in the block. To mitigate this issue, Patched frame of reference (PFOR) has been proposed [8]. This approach chooses b to be reasonable for most of the elements in the block, treating these as in FOR. The elements in a range larger than 2^b are treated as *exceptions*. In the original approach, those are stored at the end of the output, not compressed. The unused b bits are used to store the position of the next exception in the block. If b bits are not enough to store the position of the next exception, an additional one is generated. This means that one of the values is treated as an exception even if it could have been encoded using b bits.

More recent implementations of PFOR treat the exceptions differently. NewPFD [9] stores the exception positions in a dedicated area of its output, but divides them in two parts: b bits are normally stored, as for the normal values, while the remaining 32-b bits are compressed using a Simple family codec. OptPFD [9] works similarly, but chooses b to optimise the compression ratio and the decompression speed. FastPFOR [10], instead, reserves 32 different areas of its output to store exceptions. Each area contains those exceptions that can be encoded using the same number of bits. Outliers in the same exception area are then compressed using a FOR codec, to improve both the compression ratio and the decompression speed.

As discussed above, a thorough empirical study of compression codecs for realistic search engine workloads remains missing from the literature. Hence, in this paper, we experiment to address the following research question: How does the compression of (a) document ids, (b) term frequencies, (c) field frequencies and (d) term position information within inverted index posting lists affect the response times of a search engine under a realistic query load.

To address this research question, we integrate the above discussed compression codecs within a single modern IR system, and measure both the compression achieved, as well as its efficiency in retrieving a realistic set of queries, on two standard widely used research corpora. In the next section, we describe the experimental setup used to achieve this comparison.

4 Experimental Setup

We now discuss our experimental setup in terms of the IR system, corpora and queries (Section 4.1) as well as the setup of the compression codecs (Section 4.2).

4.1 IR System, Documents and Queries

We experiment using the compression codecs in Table 1 (except Golomb/Rice) by analysing their compression and decompression of inverted indices for two standard TREC corpora, namely the GOV2 and ClueWeb09 (cat. B) corpora. The GOV2 corpus of about $\sim$25M documents represents a search engine for the `.gov` domain, while the ClueWeb09 corpus of $\sim$50M Web documents is intended to represent the first tier of a general Web search engine. Each corpus is indexed using the Terrier IR platform [12], saving positions of terms in the documents and their frequencies for four fields in each document, namely title, the URL, the body and the anchor text of incoming hyperlinks. All terms have the Porter stemmer applied, and stopwords have been removed. The statistics of the resulting indices are shown in Table 2. In line with common practices [9], both corpora are reordered such that docids are assigned lexicographically by URL. Finally, d-gaps are used to represent document ids and term positions in postings for all compression codecs.

For retrieval, we use BM25 to retrieve 1000 documents using an exhaustive DAAT retrieval strategy[2]. Indeed, such a setting represents the first stage of a typical retrieval infrastructure, where additional features (e.g. proximity) would later be computed for application by a learned model [18]. For experiments on the GOV2 corpus, we use the

[2] We avoid applying dynamic pruning techniques such as WAND, to prevent skipping becoming a confounding variable within our experiments.

Table 2. Statistics about the GOV2 and ClueWeb09 corpora, with the average standard deviations ($\bar{\sigma}$) for values computed for blocks of 1024 postings

	GOV2		ClueWeb09	
	#	$\bar{\sigma}$	#	$\bar{\sigma}$
Number of documents	25,205,178	-	50,220,423	-
Number of postings	4,691,479,151	-	12,725,738,385	-
Number of positions	16,581,352,732	100.10	37,221,252,036	234.64
Number of Tokens	16,794,511,752	7.73	52,031,310,320	26.10
for title field	131,519,548	0.08	326,022,392	0.19
for URL field	426,062,666	0.05	356,811,926	0.10
for body field	15,612,059,739	5.66	26,046,648,801	4.38
for anchor field	624,869,799	2.46	11,024,667,766	23.73

first 1000 queries from the TREC 2006 Terabyte track efficiency task - these queries were sampled from a search engine query log with clicks into the `.gov` domain. For ClueWeb09, we use the first 1000 queries from the MSN 2006 query log. The average lengths of the queries in the two sets are 3.4 and 2.4, respectively.

Experiments are conducted on a dedicated, otherwise idle, 8-core (@2.13GHz) Intel Xeon processor, with 32GB RAM. The posting lists related to the query terms are loaded into main memory at the beginning of each experiment. We measure the size of the compressed inverted index, as well as the direct query response times experienced by the IR system while retrieving the query sets. In order to warm-up the execution environment, each query set is run ten times for each experiment, and the response times only measured for the last run. The mean response times for each corpus exhibited by Terrier with its baseline compression are 1.34 seconds for GOV2, and 1.55 for ClueWeb09. While these are higher than would be permissible for an interactive retrieval system, they reflect a fair comparison without resorting to distributed retrieval strategies, and without the network latencies that would then be inherent in measuring response times. Moreover, we note that while each of our postings contains four additional term frequencies for each field compared to those reported in [13], our response times are markedly faster, despite Wang et al. using the Terrier compression library.

4.2 Compression Codecs

Our baseline compression is represented by the default configuration of Terrier, which uses Gamma codec for the compression of ids and positions and Unary codec for the compression of term frequencies and field frequencies. The remaining codecs are integrated within the Terrier platform, but based on different open source implementations. As a variable byte codec, we use the Apache Hadoop[3] implementation, VInt. For classical PFOR [9, 25], we use the implementation in the Kamikaze project[4]. Finally, the JavaFastPFOR package[5] provides implementations of several codecs, namely FOR, NewPFD, OptPFD, FastPFOR and Simple 16 [26] (a member of the Simple family).

[3] `http://hadoop.apache.org/`

[4] `http://sna-projects.com/kamikaze/`

[5] `https://github.com/lemire/JavaFastPFOR`

Table 3. Average query response time μ (in seconds/query) and inverted index size β (in GB) for GOV2 and ClueWeb09. Ids and term frequencies are compressed by the shown codec; field frequencies are always compressed using Unary; positions are always compressed using Gamma. The best improvement in each column is emphasised.

	Document Ids								Term frequencies							
	GOV2				ClueWeb09				GOV2				ClueWeb09			
Name	μ	%	β	%	μ	%	β	%	μ	%	β	%	μ	%	β	%
Baseline	1.34	-	32.38	-	1.55	-	80.91	-	1.34	-	32.38	-	1.55	-	80.91	-
VInt	1.25	-6.7	35.02	8.2	1.56	0.6	84.94	5.0	1.23	-8.2	37.23	15.0	1.69	9.0	95.83	18.4
Simple16	1.24	-7.5	32.57	**0.6**	1.44	-7.1	80.78	**-0.2**	1.23	-8.2	31.65	**-2.3**	1.51	-2.6	81.50	**0.7**
FOR	1.23	-8.2	33.31	2.9	1.40	**-9.7**	81.98	1.3	1.21	**-9.7**	32.21	-0.5	1.50	-3.2	84.23	4.1
PForDelta	1.23	-8.2	33.11	2.3	1.43	-7.7	81.87	1.2	1.24	-7.5	32.29	-0.3	1.53	-1.3	83.59	3.3
NewPFD	1.22	**-9.0**	32.98	1.9	1.42	-8.4	81.30	0.5	1.22	-9.0	32.03	-1.1	1.49	**-3.9**	82.63	2.1
OptPFD	1.22	**-9.0**	32.84	1.4	1.42	-8.4	80.91	0.0	1.22	-9.0	31.98	-1.2	1.50	-3.2	82.46	1.9
FastPFOR	1.23	-8.2	33.23	2.6	1.42	-8.4	81.87	1.2	1.22	-9.0	32.08	-0.9	1.51	-2.6	83.27	2.9

To allow meaningful comparisons of compression ratios, we compress blocks of integers (document ids, term frequencies, field frequencies, positions) separately, even for bitwise oblivious codecs. In our experiments, blocks are *typically* 1024 postings, following [11]. Hence, while iterating over a posting list, our system reads a block of postings, decompresses it and consumes the postings before moving to the next block. However, for small postings lists (and the remainder of each posting list) with less than 1024 elements, block sizes can be smaller, but no less than 128 elements [9]. For blocks smaller than 128, (P)FOR codecs have to *fall back* to other codecs: we use variable byte as per [10].

In the following experiments, we consider four types of the posting list payload (ids, term frequencies, fields, positions) separately. This means that in every experiment run, only one of these payload types is encoded using the analysed compression codec, while the others remain compressed as in the baseline configuration used by Terrier, but still decoded at querying time. Experimental results for document ids, term frequencies, field frequencies and positions follow in Sections 5-8, respectively. Each section evaluates the codecs in terms of the overall inverted index size, as well as in terms of benefit to the average query response times compared to the baseline system.

5 Document ids Compression

In this section, we address the compression of the document ids, while compressing the other components of each posting as in the baseline (Unary for frequencies, Gamma for positions). The left hand side of Table 3 reports the index size (β, in GB) and mean query response times (μ, in seconds) of various compression codecs for both GOV2 and ClueWeb09. Percentage differences from the baseline compression codec (in this case, Gamma for document ids) are also shown.

On analysing Table 3, firstly, for the GOV2 corpus, we observe that each studied codec results in an inverted index larger than the baseline Gamma codec. A similar observation is obtained for ClueWeb09, where only Simple16 reduces the size of the index. Indeed, the byte-aligned VInt codec results in a 5-8% larger index. Overall, the increases in the index size are generally smaller for ClueWeb09 than GOV2. On the other hand, query response times are markedly reduced by all compression codecs (with a single exception of VInt on ClueWeb09). Indeed, using OptPFD and NewPFD, the query response times for GOV2 are reduced by 9%, while the simpler FOR reduces the query response times by 9.7% for ClueWeb09, with other PFOR-based implementations not far behind. However, this is at the cost of an increased index size ranging from ~1.3% (ClueWeb09) to ~3% (GOV2) with respect to the baseline.

Overall, in addressing the first part of our research question concerning document id compression, we find that the (P)FOR-based codecs exhibit the fastest query response times behaviour. Indeed, the efficiency of the PFOR-based codecs are as expected from [10]. However, no response time benefit for FastPFOR is observed, despite its larger index size compared to other PFOR-based codecs.

6 Term Frequencies Compression

The right hand side of Table 3 shows our results for the compression of term frequencies. Document ids, fields and positions remain as in the baseline. On analysing Table 3, we note that differently from the ids compression, it is possible to slightly reduce the inverted index size with respect to the baseline for the GOV2 corpus. In this case, the best compression (β) is obtainable using the Simple16 codec. For the FOR and PFOR codecs, index sizes are decreased (from -0.3% to -1.2% compared to the baseline), with FOR generally performing worse than any other PFOR implementations, with the single exception of PForDelta. However, in contrast, for the ClueWeb09 corpus, all compression codecs actually increase the index size compared to using Unary for the compression of term frequencies. This may be explained by the different distribution of term frequency values in the two corpora. Indeed, since ClueWeb09 is a sample of the general Web, many of its documents have a higher inlink distribution than in GOV2, with a corresponding effect on the distribution of anchor text. This leads to a greater variability of frequency values (see the $\bar{\sigma}$ columns in Table 2) and hence to a more difficult compression.

Next, in terms of average query response time (μ), for GOV2, FOR exhibits the best query response time, although other PFOR codecs are very similar, except the original PForDelta, which is slower than Simple16 and VInt. For the ClueWeb09 corpus, the benefit of using (P)FOR or Simple16 codecs to reduce the average query response time can still be observed. However, this benefit is less than that observed for GOV2. For instance, on GOV2, FOR exhibits a -9.7% reduction in average response time, but only -3.9% for ClueWeb09.

In summary, in addressing the second part of our research question concerning term frequency compression, we have observed that similar to docid compression, more advanced codecs can improve query response times (up to 9.7% faster), however, they may not result in decreased index size. Indeed, our results show that the compression of term frequencies for corpora that encapsulate higher linkage patterns (in our case, ClueWeb09) is challenging. This has not been observed by previous literature, such as [9].

Table 4. Field frequencies and positions are compressed by the codec shown in the table; document ids are always compressed using Gamma; term frequencies are always compressed using Unary. Notation as in Table 3.

	Field frequencies								Positions							
	GOV2				ClueWeb09				GOV2				ClueWeb09			
Name	μ	%	β	%	μ	%	β	%	μ	%	β	%	μ	%	β	%
Baseline	1.34	-	32.38	-	1.55	-	80.91	-	1.34	-	32.38	-	1.55	-	80.91	-
VInt	1.29	-3.7	45.69	41.1	2.55	64.5	117.85	45.7	1.32	-1.5	38.51	18.9	1.76	13.5	95.83	18.4
Simple16	1.23	-8.2	32.21	**-0.5**	1.51	-2.6	82.07	**1.4**	1.09	-18.7	35.54	9.8	1.47	-5.2	81.5	**0.7**
FOR	1.19	**-11.2**	32.82	1.4	1.47	**-5.2**	84.98	5.0	1.01	**-24.6**	33.23	2.6	1.36	**-12.3**	84.23	4.1
PForDelta	1.24	-7.5	32.74	1.1	1.53	-1.3	84.17	4.0	1.06	-20.9	34.21	5.7	1.48	-4.5	83.59	3.3
NewPFD	1.19	**-11.2**	32.73	1.1	1.48	-4.5	83.82	3.6	1.01	**-24.6**	33.35	3.0	1.37	-11.6	82.63	2.1
OptPFD	1.20	-10.4	32.71	1.0	1.50	-3.2	83.76	3.5	1.04	-22.4	32.2	-0.6	1.39	-10.3	82.46	1.9
FastPFOR	1.21	-9.7	32.75	1.1	1.49	-3.9	84.40	4.3	1.04	-22.4	31.89	**-1.5**	1.39	-10.3	83.27	2.9

7 Fields Compression

Next, we discuss the compression of field frequencies. Recall that our indices contain separate frequency counts for four different fields, namely title, URL, body and anchor text. We firstly highlight the layout of the compressed postings. In particular, two variants are plausible: (i) compression of the stream of all frequencies, regardless of field; and (ii) compression of each field separately. However, our initial experiments showed a $\sim$3% improvement in the achieved compression by the separate compression of the frequencies of each field, with a corresponding $\sim$4% improvement in response time. These results are intuitive as the distribution of frequencies across different fields are very different, as illustrated by the field statistics in Table 2 - indeed, terms occurring in URLs and titles typically have a frequency of 1, while terms occurring in anchor text and body follow Zipfian distributions, and hence exhibit markedly higher variances. For this reason, the following experiments use the same codec for each field, but separate the actual compression of each field.

The results for the compression of field frequencies are shown in the left hand side of Table 4. For the GOV2 corpus, the compression of field frequencies using modern integer compression codecs shows promising benefits in term of query response time compared to the baseline. However, this is generally at the price of a slight increase in the inverted index size. Indeed, while Simple16 can somewhat reduce the index size and VInt increases it enormously, the FOR and PFOR codecs increase the inverted index size by around $\sim$1% of the baseline. This tradeoff in index size achieves a marked decrease in response time, which ranges from -7% to -11% for the (P)FOR codecs.

For ClueWeb09, the response time benefits and compression size benefits are less marked. The FOR and PFOR codecs are between $\sim$3% and $\sim$5% faster than the baseline, with the only exception of PForDelta, which is below the $\sim$2% improvement attained by Simple16. Moreover, using a basic codec such as VInt for large corpora such as ClueWeb09 is not a feasible option: it increases the average response time by more than 50% (1.55 $\rightarrow$ 2.55 seconds in Table 4). Finally, similar to our observation for term frequencies, the compression of field frequencies on the ClueWeb09 corpus is challenging. Overall, we find that for the compression of field frequencies, the FOR codec gives the best benefit to query response time, but not the best compression.

8 Positions Compression

The right hand side of Table 4 shows the results for the compression of term positions. The compression of positions shows the most promising results. In particular, a $\sim{-25\%}$ decrease in average query response time can be attained for the GOV2 corpus along with a $\sim{-12\%}$ decrease for the ClueWeb09 corpus, using FOR. However, we note contrasting observations between GOV2 and ClueWeb09. In particular, most of the codecs attain better query response time improvements on GOV2 than on ClueWeb09. This may be explained in the higher amount of anchor text in the ClueWeb09 corpus, which is also represented in the position information of each posting. Nevertheless, there are considerable response time benefits compared to the baseline for both corpora, which is not reported in the previous literature.

We postulate that the compressibility of positions is due to the tendency of terms to occur in clusters within a document, i.e. if a term occupies a certain position in the text, there is a higher probability that it appears again soon after [7]. This suits well the list-adaptive codecs, while it is less exploitable by oblivious codecs. In fact, VInt demonstrates the smallest decrease in response time, and usually increases the index size compared to the baseline. Overall, since positions represent a considerable portion of data within a posting list (see Table 2 for the number of positions w.r.t. the number of postings in both the corpora), this explains why improvements in their compression provide marked response time benefits.

Finally, with respect to the index sizes, the compression of positions using codecs other than Gamma leads to increased space usage. However, some reductions in size are possible for the GOV2 corpus using FastPFOR or OptPFD. To summarise, in addressing the final part of our research question, we find that FOR or NewPFD represent good codecs for low query response times.

9 Conclusions

In this paper, we experimented upon two large corpora using a realistic query load to determine the query response time benefits of a range of modern integer compression codecs directly integrated within an IR system. Our thorough experiments addressed different types of posting payload information: document ids, term frequencies, field frequencies and term positions. Our findings allow a realistic estimate of the actual contribution, in term of query response time, when using compression for the different types of payload within a posting list. For instance, we highlighted the importance of the appropriate compression of term positions in benefiting query response times. Moreover, we investigated the role of anchor text in achieving high index compression and reduced query response times. In general, the list-adaptive codecs demonstrated a good tradeoff in inverted index size and query response time. As our best practice recommendation, we found that the simpler Frame of Reference (FOR) [8] codec results in fast average response times for all types of posting list payload, slightly outperforming the more complex but state-of-the-art PFOR implementations. Finally, we note that the gains obtainable from the compression of the various information within a posting list are cumulative. Indeed, when using the best aforementioned codecs for each payload type, we attain an improvement of $\sim$37% and $\sim$27% in query response time, for GOV2 and ClueWeb09 respectively, at a cost of an inverted index size increase of $\sim$6% and $\sim$12% w.r.t. the default configuration of Terrier.

References

1. Dean, J.: Challenges in building large-scale information retrieval systems: invited talk. In: Proc. WSDM 2009 (2009)
2. Anh, V.N., Moffat, A.: Pruned query evaluation using pre-computed impact scores. In: Proc. SIGIR 2006 (2006)
3. Moffat, A., Webber, W., Zobel, J., Baeza-Yates, R.: A pipelined architecture for distributed text query evaluation. Inf. Retr. 10 (2007)
4. Broccolo, D., Macdonald, C., Orlando, S., Ounis, I., Perego, R., Tonellotto, N.: Load-sensitive selective pruning for distributed search. In: Proc. CIKM 2013 (2013)
5. Witten, I.H., Bell, T.C., Moffat, A.: Managing Gigabytes: Compressing and Indexing Documents and Images, 1st edn. (1994)
6. Elias, P.: Universal codeword sets and representations of the integers. Trans. Info. Theory 21(2) (1975)
7. Yan, H., Ding, S., Suel, T.: Compressing term positions in web indexes. In: Proc. SIGIR 2009 (2009)
8. Zukowski, M., Heman, S., Nes, N., Boncz, P.: Super-scalar RAM-CPU cache compression. In: Proc. ICDE 2006 (2006)
9. Yan, H., Ding, S., Suel, T.: Inverted index compression and query processing with optimized document ordering. In: Proc. WWW 2009 (2009)
10. Lemire, D., Boytsov, L.: Decoding billions of integers per second through vectorization. Software: Practice and Experience (2013)
11. Delbru, R., Campinas, S., Samp, K., Tummarello, G.: Adaptive frame of reference for compressing inverted lists. Technical Report 2010-12-16, DERI (2010)
12. Ounis, I., Amati, G., Plachouras, V., He, B., Macdonald, C., Lioma, C.: Terrier: A High Performance and Scalable IR Platform. In: Proc. OSIR 2006 (2006)
13. Wang, L., Lin, J., Metzler, D.: Learning to efficiently rank. In: Proc. SIGIR 2010 (2010)
14. Shurman, E., Brutlag, J.: Performance related changes and their user impacts. In: Velocity: Web Performance and Operations Conference (2009)
15. Broder, A.Z., Carmel, D., Herscovici, M., Soffer, A., Zien, J.: Efficient query evaluation using a two-level retrieval process. In: Proc. CIKM 2003 (2003)
16. Cambazoglu, B.B., Zaragoza, H., Chapelle, O., Chen, J., Liao, C., Zheng, Z., Degenhardt, J.: Early exit optimizations for additive machine learned ranking systems. In: Proc. WSDM 2010 (2010)
17. Wang, L., Lin, J., Metzler, D.: A cascade ranking model for efficient ranked retrieval. In: Proc. SIGIR 2011 (2011)
18. Macdonald, C., Santos, R.L., Ounis, I., He, B.: About learning models with multiple query dependent features. Trans. Info. Sys. 13(3) (2013)
19. Williams, H.E., Zobel, J.: Compressing integers for fast file access. The Computer Journal 42 (1999)

20. Scholer, F., Williams, H.E., Yiannis, J., Zobel, J.: Compression of inverted indexes for fast query evaluation. In: Proc. SIGIR 2002 (2002)
21. Golomb, S.: Run-length encodings. Trans. Infor. Theory 12(3) (1966)
22. Rice, R., Plaunt, J.: Adaptive variable-length coding for efficient compression of spacecraft television data. Trans. Communication Technology 19(6) (1971)
23. Anh, V.N., Moffat, A.: Inverted index compression using word-aligned binary codes. Inf. Retr. 8(1) (2005)
24. Goldstein, J., Ramakrishnan, R., Shaft, U.: Compressing relations and indexes. In: Proc. ICDE 1998 (1998)
25. Zhang, J., Long, X., Suel, T.: Performance of compressed inverted list caching in search engines. In: Proc. WWW 2008 (2008)
26. Zhang, J., Suel, T.: Efficient search in large textual collections with redundancy. In: Proc. WWW 2007 (2007)