Sigma Encoded Inverted Files

Andrew Trotman
Department of Computer Science
University of Otago
Dunedin, New Zealand
andrew@cs.otago.ac.nz

Vikram Subramanya
Department of Computer Engineering
National Institute of Technology Karnataka
Surathkal, India
vicky.nitk@gmail.com

ABSTRACT
Compression of term frequency lists and very long document-id lists within an inverted file search engine are examined. Several compression schemes are compared including Elias γ and δ codes, Golomb Encoding, Variable Byte Encoding, and a class of word-based encoding schemes including Simple-9, Relative-10 and Carryover-12. It is shown that these compression methods are not well suited to compressing these kinds of lists of numbers. Of those tested, Carryover-12 is preferred because it is both effective at compression and fast at decompression.

A novel technique, *Sigma Encoding* prior to compression, is proposed and tested. Sigma Encoding utilizes a parameterized dictionary to reduce the number of bits necessary to store an integer. This method shows an about 0.3 bit per integer improvement over Carryover-12 while costing only about 3 extra clock cycles per integer to decompress.

Categories and Subject Descriptors
H.3.1 [Information Storage and Retrieval]: Content Analysis and Indexing - *Indexing methods*

General Terms
Algorithms, Performance

Keywords
Inverted Files, Compression

1. INTRODUCTION
Inverted file search engines are thought to be I/O bound. The amount of time it takes to process the index and produce the list of results is more a function of the time it takes to read the postings from disk than the time to process the postings. For this reason, Zobel & Moffat [7] and Williams & Zobel [6] recommend adding compression in order to increase throughput.

Scholer *et al*. [4] and Trotman [5] examine two fundamentally different forms of index compression: bit-based and byte-based. Trotman [5] compares the effectiveness and decompression speed of Elias-γ Elias-δ, Golomb, Binary Interpolative Coding, and Variable Byte Encoding and shows that Variable Byte Encoding requires more storage, but can be decompressed more efficiently than others. Scholer *et al*. [4] showed that a search engine using Variable Byte Encoding is more efficient that one based on bit-wise compression schemes.

Variable Byte Encoding is preferred because the rate of decompression easily compensates for the loss in compression effectiveness. Decompression can be performed in as little as 10% of the time taken for bit-based schemes such as Golomb. The loss in storage space, however, can be as much as 300%.

Anh and Moffat [2] propose several word-based compression schemes. These schemes pack many integers into a fixed-sized word by carving that word into a number of fixed bit-length pieces. Doing this allows many integers to be packed into a word, whereas Variable Byte Encoding packs at most one integer into a byte.

Several schemes have been proposed including Simple-9. Simple-9 compression uses a 32-bit word divided into two parts, a 4-bit selector and a 28-bit body. This 28-bit body can be divided into 9 different fixed-sized chunks (1, 2, 3, 4, 5, 7, 9, 14, and 28 bits). The 4-bit selector is used to describe which division is being used. Decompression is fast and compression effectiveness is high.

Relative-10 compression shrinks the selector to just 2 bits (using 30 bits for storage). Instead of directly representing the division, it represents the division relative to the previous word. That is, the next word might be divided the same way, one worse, or one better then the current word (or it might be reset to one integer only).

When the division of 30 bits into 7-bits each (or 4-bits each) is performed, there are 2 wasted bits in a word. These might be used as the selector bits for the next word (consequently 32 bits are available in the latter). Exactly this is the case in Carryover-12 compression.

The algorithms of Anh and Moffat [2] result in compression almost as effective as bit-wise schemes while being almost as efficient as Variable Byte Encoding at decompression.

Common to these prior studies is the desire to increase throughput and decrease the storage space necessary to store the document-ids in an inverted file search engine. We, instead, examine compression of the term frequencies. We show that the nature of the term frequencies is different from that of document-ids and

CIKM'07, November 6--8, 2007, Lisboa, Portugal.

propose a dictionary-based encoding (Sigma Encoding) followed by integer compression. When used with Carryover-12, a space saving of about 0.3 bits per integer for an extra about 3 cycles per integer to decompress is shown as compared to Carryover-12 without Sigma Encoding.

2. INVERTED FILE INDEXING

The postings for a single term in an inverted file search engine are often represented as: $<df_t: <d_{t1},tf_{t1}>, <d_{t2},tf_{t2}>, \ldots, <d_{tn},tf_{tn}>>$ where df_t is the number of documents containing term t (the document frequency) and tf_{tn} is the number of times the term occurs in document d_{tn} (the term frequency). The df_t component is often stored in the vocabulary and the $<d_{tn},tf_{tn}>$ (posting) pairs in the postings file.

Investigations into ways of storing postings suggest the document-ids and the frequencies should be encoded separately and stored sequentially [3]. This allows the loading of only document-ids for Boolean searching, or additionally the frequencies for ranking.

The document-ids are a monotonic sequence and delta coding is often applied. The sequence $<d_1, d_2, d_3,\ldots, d_{tn}>$ for example <3, 10, 16, …, 123> is stored as $<d_1\text{-}0, d_2\text{-}d_1, d_3\text{-}d_2, \ldots, d_{tn} - d_{tn-1}>$ for example <3, 7, 6, …, 1>. The deltas (or differences) are, by necessity, smaller than the ids and hence compress more effectively with adaptive compression. As the frequency of a term increases, the average delta must decrease in size. The deltas for a term occurring in every document form the sequence <1, 1, 1, …>, with a mean of 1. For a term occurring in half the documents, the mean delta will be 2.

Term frequencies, the tf_{tn} component of the postings, unlike document-ids, do not form a monotonic sequence. Further, as document frequency (df_t) increases, we expect the average term frequency (tf_{tn}) to increase as well, and so we expect the sequence to compress less effectively. The grammatical conjunctions (but, when, etc.), for example, are expected to occur in a large number of documents and to occur many times in those documents. The consequence of this observation is that as document frequency increases, the document-ids will compress more effectively, but the term frequencies will compress less effectively.

2.1 Impact ordering

Impact ordered inverted files [1] are represented $<i_1:d_{i1\text{-}1}, d_{i1\text{-}2}, d_{i1\text{-}3},\ldots, d_{i1\text{-}tn}, i_2:d_{i2\text{-}1}, d_{i2\text{-}2}, d_{i2\text{-}3},\ldots, d_{i2\text{-}tn}>$ where i_j is the impact factor of the given set of documents. The impact factor is, essentially, a coarse-grained bucketing of term frequency. With respect to compression, documents with the same impact factor form a monotonic sequence.

Using impact factors does not fundamentally change the nature of inverted file compression; however it does (essentially) remove the necessity of compressing term frequencies. Section 4 demonstrates that Sigma Encoding is effective for very long lists of document-ids. These continue to be seen even with impact ordering; however, we leave for further work the demonstration of the effectiveness of Sigma Encoding in impact ordered inverted files.

3. SIGMA ENCODING

Each term frequency list is compressed separately and there are no dependencies between the lists. Given a single list, a dictionary is constructed for that list; it is a dictionary of the unique values seen in the list. The dictionary is then sorted by decreasing frequency of occurrence of value in the list. For example, for <5, 3, 12, 5, 3, 5, 5, 3, 12, 4>, the most frequent value is 5, then 3, then 12 and finally 4, so the dictionary is <5, 3, 12, 4>. Integers in the original list are then renumbered with the ordinal value from the dictionary (the *sigma*). In the example, this results in the sequence <0, 1, 2, 0, 1, 0, 0, 1, 2, 3>, when counting from 0. This is standard dictionary-based compression.

To encode each term frequency list, it is necessary to encode the dictionary length, the dictionary, and the sigmas. In the example, it is necessary to store <<3><5, 3, 12, 4><0, 1, 2, 0, 1, 0, 0, 1, 2, 3>>. This new sequence is longer than the original list. But if an adaptive compression scheme such as Carryover-12 is used to further compress the whole sequence, gains can be seen because the most frequent terms are represented by the smallest sigmas, which in turn compress well.

3.1 Dictionary Compression

Carryover-12 encodes integer i in $\log_2(i)$ bits (at best). It is, consequently, possible to swap two entries in the dictionary with no effect on the number of bits used to compress a sequence containing the two, as long as their sigmas are of the same magnitude. That is, if a dictionary entry is in position 14 and another in position 15, each takes 4 bits to store, so the order of the two does not matter when considering how many bits are necessary to store the sigma (because both sigmas are 4 bits in length). Reordering the dictionary is effective if, consequently, the dictionary can be stored more efficiently.

Within the dictionary, all values whose sigmas are of the same base-2 magnitude (take the same number of bits to represent) are sorted into increasing order. To compress the dictionary, delta coding is applied to each base-2 range, and then compression using another compression scheme (such as Carryover-12).

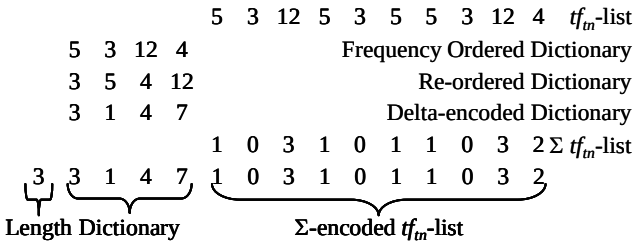

Figure 1: Sigma Encoding. First, a frequency-ordered dictionary is created, which is then ordered and delta-coded. The dictionary length, the dictionary and the sigmas are combined. This list is later compressed with an adaptive compression scheme such as Caryover-12

Figure 1 illustrates the compression process for a single term frequency list. First, the frequency ordered dictionary is constructed, then those entries that take 1-bit to store (0-1), and then 2-bits to store (2-3) are sorted into increasing order. To each of these ranges delta encoding is applied. The tf_{tn}-list is renumbered with the sigmas. Finally, the entire sequence is composed of the dictionary length ($|D|$), the delta-encoded dictionary, and sigma-encoded term frequency values. Of course, lengths, deltas and sigmas count from 0 (sigma=0 is the first dictionary entry).

There exists the possibility that every term in the original list is unique. In this case the dictionary doubles the length of the sequence. This can result in Sigma Encoding adding an overhead. Such a problem is likely to occur when the lists are short, which is frequently in an inverted file index.

To alleviate this, we parameterize the scheme. Only terms occurring more than threshold T times are added to the dictionary, the others are stored as $|D| + tf_{tn}$ where $|D|$ is the dictionary length. Throughout this investigation, we set T=1, which means that a value must occur two or more times to enter the dictionary.

When indexing very large document collections, the postings for a single term could become very large and contain many distinct values. If this is the case, the dictionary could also become very large. The size of the dictionary might be controlled either by only taking terms that occur more than some T number of times, or else by imposing a strict limit on the length.

4. EXPERIMENTS

4.1 Experimental Environment

Experiments were conducted on an Intel Celeron processor at 1.06GHz. The document collection was the TREC Wall Street Journal (WSJ) Collection. Further experiments used the TREC Wt10g web collection and a Pentium 4 2.8GHz processor. Results for Sigma Encoding include the cost of storing and decompressing the dictionary (one for each list).

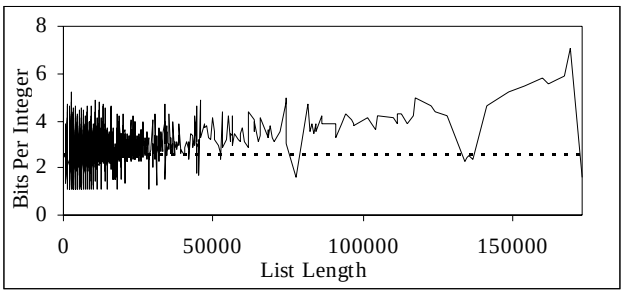

Figure 2: The effectiveness of Carryover-12 compression on term frequency lists (in bits per integer) decreases as the length of the list (document frequency) increases

4.2 Experiment 1

We tested the hypothesis that the compression effectiveness of term frequency lists decreases as document frequency increases.

The postings lists were generated for each unique term in the Wall Street Journal collection. They were then compressed using Carryover-12. In Figure 2 the mean number of bits per integer (BPI) needed to store the term frequency lists is plotted against increasing document frequency (the mean is shown dotted).

Although unstable at the beginning, a general upward trend is seen suggesting that, indeed, compression effectiveness of term frequency lists decreases as the number of documents in which the term is seen increases.

Table 1: Compress effectiveness and decompress efficiency of bit-wise, byte-wise, word-wise compression schemes and for Sigma Encoded Carryover-12. Values are shown for the TREC Wall Street Journal Collection

BPI	δ	γ	Gol	Byte	S-9	R-10	C-12	$\Sigma_{T=1}$
tf_{tn}	4.5	3.4	2.7	8.0	3.1	3.1	3.1	2.8
d_{tn}	8.7	8.5	6.2	9.4	7.6	7.2	7.0	7.3
CPI								
tf_{tn}	65.1	44.7	42.6	9.0	13.2	13.9	14.4	17.0
d_{tn}	113.8	102.5	99.5	11.5	15.0	16.9	16.7	20.7

Table 2: Compression effectiveness and decompression efficiency of Carryover-12 and Sigma Encoded Carryover-12. Values are shown for the TREC Wt10g

	BPI C-12	BPI $\Sigma_{T=1}$	CPI C-12	CPI $\Sigma_{T=1}$
tf_{tn}	3.7	3.5	21.2	25.2
d_{tn}	8.3	8.8	25.4	33.8

4.3 Experiment 2

Compression effectiveness and decompression efficiency for both term frequencies and delta-encoded document-ids were compared to several other schemes. Table 1 shows (left to right) effectiveness in bits per integer (BPI), of Elias-δ, Elias-γ, Golomb, Variable Byte Encoding, Simple-9, Relative-10, Carryover-12, and Sigma Encoded Carryover-12. For term frequency lists Sigma Encoding shows an improvement on Carryover-12 of 0.3 bits per integer. The additional cost of decompression in clock-cycles per integer (CPI) is about 3.

A comparison of Carryover-12 with Sigma Encoded Carryover-12 in the larger TREC Wt10g collection is presented in Table 2. There an improvement of 0.2 bits per integer is shown for an additional cost of 4 clock cycles per integer.

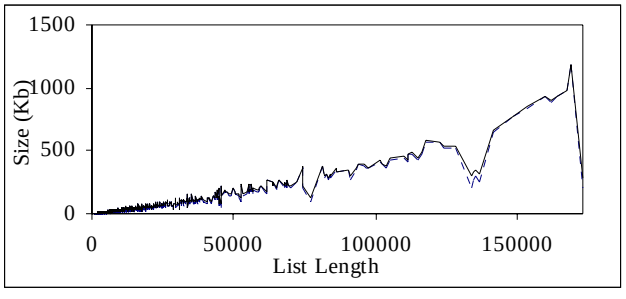

Figure 3: Comparison of Sigma Encoded Carryover-12 (dashed) to Carryover-12 compression on term frequency lists as document frequency increases

4.4 Experiment 3

Using the Wall Street Journal collection, the performance of Sigma Encoded Carryover-12 was compared to Carryover-12 as document frequency increases. In Figure 3 the solid line shows the size of the compressed list when Carryover-12 is used, the dashed line shows the same for Sigma Encoded Carryover-12. It can be seen that the improvements are small but consistent.

The same comparison was performed for document-id lists where encoding proved ineffective when averaged over all lists (see Table 1 and Table 2). The results are presented in Figure 4 where it can be seen that Sigma Encoding is effective when document frequency is large, but not so when small. For long lists, a more ordered distribution around the mean is expected (due to the standard error of the mean) and hence, the dictionary is more effective.

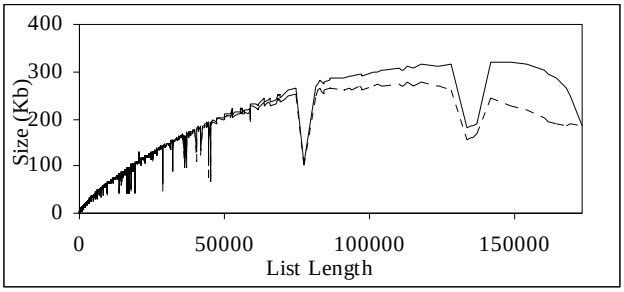

Figure 4: Comparison of Sigma Encoded Carryover-12 (dashed) to Carryover-12 compression on document-id lists as document frequency increases

5. CONCLUSIONS AND FUTURE WORK

Compression effectiveness for term frequencies was examined and (at least in the Wall Street Journal collection) the effectiveness in bits per integer was shown to decrease as the document frequency increases. This is likely to be because terms that occur in many documents are terms that are likely to be frequent in a single document (such as the grammatical conjunctions and articles). The larger term frequency values compress less well than the smaller term frequency values seen for uncommon terms.

A dictionary-based coding scheme called Sigma Encoding is introduced as a preprocessing step before compression. This coding scheme, when used in conjunction with Carryover-12 compression, is shown to be effective in compressing term frequency lists.

When tested on document-id lists, Sigma Encoding followed by Carryover-12 compression is shown to be ineffective in the general case, but effective when document frequencies are large (the lists are long).

In future work we plan to examine Sigma Encoding with impact ordered indexes. We expect it to continue to be effective; perhaps more so than on non-impact ordered lists. For the simple case of two impacts, the monotonic document-id list is represented by two shorter monotonic lists, one for each impact. The mean delta in each list will necessarily be larger than the single list; therefore, dictionary compression can be expected to be effective in reducing the number of bits necessary to store each delta. With more than two lists, the gaps will become larger and effectiveness is expected to be increase.

We also plan to examine Sigma Encoding with phrase searching. In this case, the gaps between term occurrences are expected to be large, but may not form regular patterns.

The effectiveness of Sigma Encoding is dependant on how well the dictionary can be stored. We are examining techniques to store the dictionary more efficiently, especially when long sequences of consecutive numbers are present. We are also examining alternative threshold methods.

Sigma Encoding followed by Carryover-12 compression is an effective method of storing term frequency lists and long document-id lists in an inverted file search engine. An improvement of about 0.3 bits per integer is seen at the cost of about 3 clock cycles to decompress. Further improvements are expected with an improved threshold method and better compression of the dictionary.

6. REFERENCES

[1] Anh, V. N., & Moffat, A. (2002). Improved retrieval effectiveness through impact transformation. *Australian Computer Science Communications*, 24(2), 41-47.

[2] Anh, V. N., & Moffat, A. (2005). Inverted index compression using word-aligned binary codes. *Information Retrieval*.

[3] Anh, V. N., & Moffat, A. (2006). Structured index organizations for high-throughput text querying. In *Proceedings of the 13th Int. Symp. String Processing and Information Retrieval*, (pp. 304-315).

[4] Scholer, F., Williams, H. E., Yiannis, J., & Zobel, J. (2002). Compression of inverted indexes for fast query evaluation. In *Proceedings of the 25th ACM SIGIR Conference on Information Retrieval*, (pp. 222-229).

[5] Trotman, A. (2003). Compressing inverted files. *Information Retrieval,* 6(1), 5-19.

[6] Williams, H. E., & Zobel, J. (1999). Compressing integers for fast file access. *Computer Journal,* 42(3), 193-201.

[7] Zobel, J., & Moffat, A. (1995). Adding compression to a full-text retrieval system. *Software - Practice and Experience,* 25(8), 891-903.