Using Inverted Files to Compress Text

Strahil Ristov
Ruđer Bošković Institute, Zagreb, Croatia

This is the first report on a new approach to text compression. It consists of representing the text file with compressed inverted file index in conjunction with very compact lexicon, where lexicon includes every word in the text. The index is compressed using standard index compression techniques, and lexicon is compressed by original dictionary compression method that gives better compression results than existing procedures. Compression procedure is complex, but decompression time is linear with the file size, although it requires two passes and hence can not be performed online. First experiments show that this method, when refined, can be competitive for larger texts that only need to be decompressed in the real time.

Keywords: text compression, inverted file, index compression, lexicon compression.

1. Introduction

Data compression is one of the most widespread application in computer technology. Algorithms and methods that are used depend strongly on the type of data, i.e. whether the data is static or dynamic, and on the content that can be any combination of text, images, numeric data or unrestricted binary data. There are various algorithms for data compression, some for general purposes and some fine-tuned for specific data types. Their performance is measured by three factors: compression ratio, compression time and decompression time. Compression ratio is on the whole the most important factor, but for general purpose algorithms both compression and decompression times must be fast. In applications where data is compressed once and widely distributed in compressed form, compression time is less important, only decompression time should be fast. A comprehensive survey of existing algorithms can be found in [1], [4].

An important part of the data compression effort goes to text compression. The quantity of text produced in electronic format in the world grows fast, and a lot of it has to be kept in storage for long time. Therefore, a quest for better text compression method is always warranted. There are many instances where text is compressed in advance and distributed widely (e.g. CD distribution). In such cases compression time is not important (as long as it is reasonable) since the data can be compressed once on a powerful computer and used later on a number of weaker machines.

Here we propose a method for compressing texts that seems to be promising for large text files that need only to be decompressed in real time. The compressed file is a su ccinct representation of the inverted file and decompression process is actually the reconstruction of the original file from the inverted one. Decompression procedure takes time liner with the size of the output file.

The paper is organized as follows: In section 2 we describe inverted file representation, in section 3 we present the algorithms used for compacting index and lexicon and in section 4 we present some preliminary results on actual data. We conclude with the discussion in section 5.

2. Inverted files

Inverted file is the most widely used structure for indexing and accessing data in document collections. It is a list of pointers to each occurrence, or an index, of each term in the document collection [4]. The list of terms that are indexed is called lexicon and consists of terms that are regarded as interesting. Each term in lexicon has the associated pointer to the beginning of the list

of pointers to term occurrences. These pointers are multidimensional, as they must usually designate the document, chapter and paragraph in which the term occurs. Inverted file is the single most efficient method for accessing the data. However, we feel that the potential of using inverted file indexing for compressing texts has been unjustifiably neglected.

If we consider only one text file and use distinct words as lexicon terms, the pointers can be one-dimensional, just the positions of each word in the file. An example of this is presented on Fig. 1 for the extraordinarily structured text (borrowed from [4]) of a children nursery rhyme:

Peas porridge hot,
Peas porridge cold,
Peas porridge in the pot
Nine days old.

Some like it hot,
Some like it cold,
Some like it in the pot
Nine days old.

Term	Positions	Δ coded positions
Peas	1, 4, 7	1, 3, 3
Porridge	2, 5, 8	2, 3, 3
Hot,	3, 18	3, 15
Cold,	6, 22	6, 16
In	9, 26	9, 17
The	10, 27	10, 17
Pot	11, 28	11, 17
Nine	12, 29	12, 17
Days	13, 30	13, 17
Old.	14, 31	14, 17
Some	15, 19, 23	15, 4, 4
Like	16, 20, 24	16, 4, 4
It	17, 21, 25	17, 4, 4

Fig. 1. An example of the inverted file representation.

The list of terms from the nursery rhyme is presented in the first column of the table in Fig 1. To include every character in the file we can define terms as the content between two spaces or new-line characters. This actually produces terms that are distinct words, sometimes with punctuation attached. Positions of each term in the file are listed in ascending order in the second column of Fig 1. For a complete definition of the inverted file the information on how many occurrences exist of a given term should be included. This can be done by explicitly stating that number at the beginning of the list for each term, or by using an escape value as the last element in the list, or by adding an extra signal bit to all the values in the list.

3. Compressing inverted files

We consider now the possibilities for reducing the size of the index and of the lexicon. The issue of compressing inverted file has been studied extensively and the best methods for the index compression are very efficient. On the other hand, by using our own lexicon compression methods, we were able to improve considerably the compression performance of the hash table that is regarded as the best structure for keeping the lexicon in inverted file data access system. In the next two sections we describe procedures for index and lexicon compression that we applied in our experiments.

3.1. Compressing position index

If positions values are kept explicitly, not much can be done in the way of compression. If there are T terms in the file then each number from 1 to T appears only once in the inverted file and if the ordering is random the least amount of the space needed for the full position list is $T \times \log T$. However, if position list for each term is ordered, then all values except the first one can be delta (Δ) coded, i.e. represented as the difference from the preceding value. In this way distribution of the numbers in the inverted file is less uniform, many values occur more than once, some with higher frequencies than others. This makes it possible to use statistical compressing procedures where more frequent symbols are represented with shorter codes.

Delta coded position lists for the nursery rhyme example are presented in the third column in Fig 1. Evidently, some values are more frequent than others and the frequency spread is from 7 (for value 17) to 1. Now, some statistical coding can reduce overall index size considerably. According to [4], for general purposes the most efficient way to compress a list of numbers with skewed distribution of frequencies is to use local

Bernoulli method for describing the probability distribution in conjunction with Golomb coding technique. For the purposes of this work we used simpler global canonical Huffman coding method. Frequencies for each value were taken from the whole file and used to generate canonical Huffman codes. Obviously, somewhat better result should be expected if term frequencies are taken locally for each term, or for a batch of terms. In this preliminary work we opted for the simpler method that we already used successfully in [3] for complex phonetic lexicon compression.

3.2. Compressing lexicon

In standard inverted file applications the lexicon must enable random access where lexicon query for each term returns the pointer to the list of its positions. However, if there are no queries and the only operation a lexicon should support is to list its content in given order then it is possible to represent it in a very compact way. It is exactly on this fact that we base our expectations that inverted file can be efficiently used for text compression.

If terms are natural language-based, the most space efficient representation of the lexicon is a compressed trie. Trie, or digital search tree, is a tree with only one input symbol per node. When such a tree is constructed for a set of strings, prefixes are shared and search procedure is very quick. It branches at each symbol of input word needing only L comparisons for a word of L symbols.

For natural language data digital trees are sparse and we use linked lists representation for nodes to eliminate empty transitions. Then linked list trie can be compressed by replacing repeated sub-lists with pointers to their first occurrences. When this procedure is fine-tuned, resulting compressed trie is very compact [2], in fact, to our knowledge, it's the most compact representation of a set of strings. The best results we were able to obtain for huge word lists in various languages are 2 bits per word for one million Croatian words, 4 bits per word for 600.000 French words and 18 bits per word for 350.000 English words. This applies to extreme, dictionary type word lists. Experiments, however, showed that for much less comprehensive word lists only a few–bits–per–word increase can be expected.

3.3. Time and space complexity

Compression process is complex, it should go like this: 1) text must be parsed for terms, 2) term frequencies and term positions must be found, 3) position lists must be delta coded and then new values must be replaced with Huffman codes, 4) lexicon of terms must be sorted (that is required by our trie compression algorithm [2]) and then trie must be produced and compressed. Detailed analysis shows that time complexity is bounded by O (S + T log T) where S is the size of text file and T is the size of lexicon of terms. Although this is only by log T slower than linear, the constants are important and compression process apparently takes too much time if it has to be done in real time.

Decompression, on the other hand, can be performed considerably faster in O (S + TO log TO) time, where TO is the number of term occurrences in the file. Two passes are needed so online decompression is impossible, but the speed is satisfactory for general purpose applications. Decompression is performed in the following manner: All terms from the lexicon are output to auxiliary memory space and the addresses for each term are stored in the term address table. Then all term position lists are output to two-element table where one element is the position and the other is the number of the term. The table is sorted (this is the step that requires TO log TO time) by the position value and finally terms are printed out following the order in the sorted table. Alternatively, if there is enough RAM available, one element table of size TO that would hold only term ordinal numbers can be used. Then, as position lists are decompressed, the table is filled, with term number at the position of the value in the list. After the table is completely filled, the terms are output following the table ordering. This way the complexity is linearized to O (S + T).

4. Experimental results

We tested this method on Canterbury Corpus' (a reference data set for compression algorithms, http://corpus.canterbury.ac.nz/) two text files:

play and *text* file. The *text* is Lewis Carroll's Alice in the Wonderland, and the *play* is Shakespeare's As You Like It. These are both relatively small texts and results are not spectacular. The best results can be expected with large texts, or text collections processed together. We wanted to see how compression performance improves by adding more texts of similar nature so we did the experiment with four Shakespeare's plays taken from Project Gutenberg collection (http://promo.net/pg/). First we compressed As You Like It, and then added three more plays: Hamlet, A Midsummer Night's Dream and Richard III.

In Table 1 results for two files from Canterbury Corpus are presented. Compression ratio is given in the number of bits needed to represent one character. We included values for the best overall compression reported, and for the widespread gzip utility (used with '-best' option) in order to give some perspective on our results. In Table 2 results are presented for one and four plays from Project Gutenberg. Here we included the number of different terms and the total number of term occurrences to demonstrate how the number of terms grows considerably slower than the text size, resulting in better compression for larger texts.

	Play	Text
File size	125179 byte	152089 byte
Inverted file compression	3.69	3.19
Best compression	2.51	2.24
gzip-9	3.12	2.85

Table 1. Compression results for two files from Canterbury Corpus.

	As You Like It (PG)	Four plays (PG)
File size	140116 byte	650126 byte
Number of distinct terms	5395	17270
Total number of term occurrences	23187	104741
Inverted file compression	3.35	3.05
gzip-9	2.91	2.91

Table 2. Compression results for one and four Shakespeare's plays from Project Gutenberg collection.

5. Discussion

The complexity of implementation and low speed makes the inverted file compression method competitive only if compression ratio is better than with other methods. The best results achieved so far for English text are around 2.2 bits per character. The question is, can this method perform better? As expected, effectiveness of inverted file compression method improves considerably with larger texts. In our experiments this was due to better lexicon compression with larger number of terms. Compressed lexicon for As You Like IT (PG) takes 26.0 bits per term, and 22.7 bits per term for Four plays (PG). There are 4.3 times more term occurrences in As You Like It (PG) than distinct terms, so only 6.0 lexicon bits are needed for each term occurrence. This values for Four plays are 6.1 and 3.7.

In both cases the number of bits necessary for storing the position of one term occurrence is about 15. The total number of bits needed to store one term occurrence, or one word in the text, is about 20 for one play and 18 for four plays. The average word size is 6 characters, therefore each word should be represented with under 13 bits if we want to improve on existing results. This might be possible to achieve in two ways, one is to fine tune index compression - the best results in index compression are reported to be about 6 bits per index entry so there should be place for improvement from our 15 bits.

The second is to find better assignment of lexicon terms. In our experiments everything between two spaces is regarded as a term. That way, even articles have their own index position entries, which is clearly uneconomical. If articles are concatenated with adjacent words to form compound terms, all article positions are omitted from position lists, and the increase in lexicon size is marginal. This can be extended to other frequent word pairs or sequences. One can use suffix tree to find most frequent word sequences in the text and build the term lexicon according to that. We expect that further

work along these lines will prove inverted file text compression to be a competitive method for large texts in applications where compression time is not critical.

References

[1] BELL T. C., CLEARY J. G., WITTEN I. H., *Text Compression*. Prentice-Hall, 1990.

[2] RISTOV S., LAPORTE E., Ziv Lempel compression of huge natural language data tries using suffix arrays. In: Crochemore M, Paterson M, editors. *Proceedings of the 10th Annual Symposium Combinatorial Pattern Matching, CPM'99*, July 1999, University of Warwick, LNCS 1645, 1999. pp. 196–211.

[3] RISTOV S., LAPORTE E., A Method for Compressing Lexicons, *Data Compression Conference DCC02*, April 2002, Snowbird, UT, USA. pp. 470.

[4] WITTEN I. H., MOFFAT A., BELL T. C., *Managing Gigabytes*. Morgan Kaufmann, 2nd edition, 1999.

Received: June, 2002
Accepted: September, 2002

Contact address:
Strahil Ristov
Ruđer Bošković Institute
Bijenička 54
Zagreb, Croatia
e-mail: ristov@rudjer.irb.hr

Strahil Ristov is associated scientist at Ruđer Bošković Institute, Department of Electronics. He received his B.S., M.S. and Ph.D. degrees from the Faculty of Electrical Engineering and Computing, University of Zagreb. His research interests include string algorithms, data compression and computational linguistics.