Buletinul Ştiinţific al Universităţii "Politehnica" din Timişoara

Seria ELECTRONICĂ şi TELECOMUNICAŢII
TRANSACTIONS on ELECTRONICS and COMMUNICATIONS

Tom 49(63), Fascicola 2, 2004

On the Performances of Symbol Ranking Text Compression Method

Radu Rădescu, Răzvan Popa[1]

Abstract – This paper presents an implementation of a method for text compression, first described by Shannon in 1951 and later, in 1997, by Fenwick. Unlike other compressors, which exploit symbol frequencies in order to assign shorter codes to more frequent symbols, this technique prepares a list of probable symbols to follow the current one, ordered from most likely to least likely. Tests were conducted on various input sources and the results are shown here. The implementation supports further optimizations, as it will be explained.

Keywords: symbol ranking, context, number of tries

I. INTRODUCTION

Symbol ranking compression is a method initially described by Shannon in 1951 [1] and later, in 1996, by Fenwick [2]. The algorithm is expecting to encounter repeating strings in the input, which makes it more suitable for repetitive text, such as human language. It consists of two blocks: a seeking function, for both compression and decompression, which is used to suggest a symbol, and a processing function, specific to the action being carried out.

Other similar methods do exist, by Bentley et al. [3], Howard and Vitter [4] or Burrows and Wheeler [5] (the Burrows-Wheeler Transform), but with little reference to Shannon's original work. BWT could be regarded as a symbol ranking compressor, with the Move-To-Front list acting as a good estimate of symbol ranking.

II. ALGORITHM DESCRIPTION

The algorithm extends one of Charles Bloom's methods of offering possible symbols in the approximate order of probability of their occurring in the current context. Bloom [6] noted that the longest earlier context, which matches the current context, is an excellent predictor of the next symbol. However, this implementation only uses contexts of a certain maximal length, as it would be more time-consuming to leave the context unbound.

The first block of the two mentioned above is suggesting which the next symbol could be, based on the current context. It starts by searching for a string of the maximum permitted length matching the current context and as soon as it is found, the next symbol is offered. If the offer is rejected, the search continues until there are no possible strings of this length left in the buffer. Then the length is decremented and the search starts over. The context's length can go as low as 1, meaning a symbol could be the context, but no lower. When a string is found to be equal to the current context, the symbol next to it is first checked to see if it was not offered before and rejected, in which case it is not used again.

The second block performs as a validation. It reads a symbol and asks for suggestions, a kind of guess game. If it receives a good answer, it outputs a "1" bit, otherwise a "0" bit: this is where the compression occurs. It does not accept more than a certain number of guesses, if, until that, the right symbol has not been offered the search is aborted, the correct symbol is output, and the scheme moves on. On decompression things work quite similar, but now the answers are read from the compressed file, be it a "1" or a "0" bit, or the correct symbol.

III. EXPERIMENTAL SOLUTION

A software implementation of this compression method was performed. The program uses a circular buffer to keep track of processed text, both on compression and decompression. This means that when the first symbol is read it is inserted at the first address in the buffer, and subsequent symbols follow it, until the end of the buffer is meet, then symbols are again inserted from the first position, and so on.

Obviously, this way some possible better contexts in the past are lost, and an ideal approach would store in the memory all the previous text, instead of a bound length buffer. That is why the buffer's length can be varied to some extent, to try to accustom to different input texts.

[1] Facultatea de Electronică şi Telecomunicaţii, Catedra de Electronică Aplicată şi Ingineria Informaţiei, Bd. Iuliu Maniu nr. 1-3, sector 6, Bucureşti, e-mail: rradescu@atm.neuro.pub.ro

To find a matching context the search begins in the buffer at the previously inserted symbol with the maximum context length. It seeks backwards for a symbol that is equal to the one at the end of the context.

Having found one, it tries to match the symbol at the half of the current context to the one at the half of the possible candidate context and then the first symbols, in the context and in the candidate. If all these symbols match, then a complete string comparison is performed, and, if it succeeds, the symbol is offered and marked in the exclusion table. This approach is used to avoid unnecessary comparisons, and many false contexts fail the half- or start- symbol test.

An example of how the compression sequence could look like is presented in the following. The text to be coded is "the symbol is offered", the context is "Having found one [...]", from the previous paragraph, all in the buffer. At most 5 unsuccessful attempts are allowed. The maximum context size is 4. In Table 1, the first column is the original text and the columns to the right are the attempts of guessing the next symbol. The rightmost cell on every row contains the sequence of output symbols for the corresponding input character, underlined characters representing a binary value (a "1" bit or a "0" bit) and the numbers between parentheses – the context length at which the right symbol was found, or (n/a) if no guess was successful. White spaces were converted to underscores for reasons of readability.

Table 1

Context: Having found one, it tries to match [...]							
Text	Attempts						Output
t	i	a	t				001 (2)
h	h						1 (3)
e	e						1 (4)
_	n	s	_				001 (4)
s	c	f	p	h	o	s	00000s (n/a)
y	y						1 (4)
m	m						1 (4)
b	b						1 (4)
o	o						1 (4)
l	l						1 (4)
_	s	_					01 (4)
i	a	t	s	i			0001 (1)
s	t	f	s				001 (2)
_	p	o	y	,	u	_	00000_ (n/a)
o	p	t	i	s	a	o	00000o (n/a)
f	n	l	r	m	s	f	00000f (n/a)
f	_	o	i	f			000f na
e	_	o	i	e			000e (na)
r	_	d	e	r			0001 (1)
e	f	m	i	s	e		00001 (1)
d	n	r	_	d			0001 (1)

For the considered text, the output is:

0011100100000s1111101000100100000_00000o000
00f000f000e0001000010001.

which means that for the 168-bit input, the coder outputs 110 bits. giving a compression ratio of 0.65476. Of course, the result depends a great deal on the parameters used in the algorithm and on the context (the previous symbols).

IV. COMPRESSION RESULTS

The following tests were conducted using 5 files of various content and dimension:

- *codulpenal.txt* – Romanian text, legal;
- *book1.001* – English text, fiction book (incomplete, in order to have about the same length as the first one – a comparison between the two languages' compressibility was attempted);
- *obj1* – object code for VAX machine, binary file;
- *pic* – black & white fax picture (it is supposed to have a big redundancy and to be very compressible);
- *progl* – source code in LISP.

Every file (except the first one) is part of the Calgary Corpus collection [7]. The goal was to try different scenarios for the use of this algorithm, as it is a known fact that a compression method can yield better results only for certain file types.

The program can be tuned by three parameters:

(a) the length of the buffer:
(b) the maximum context length:
(c) the number of tries before outputting the unchanged symbol.

The following graphics represent the compression ratio of the symbol ranking method when modifying only one of the parameters, keeping the others to some fixed best-ratio values.

The columns grouped by five in the histograms below (see Fig. 1, 2, and 3) represent the compression ratio of the files in the above listed order.

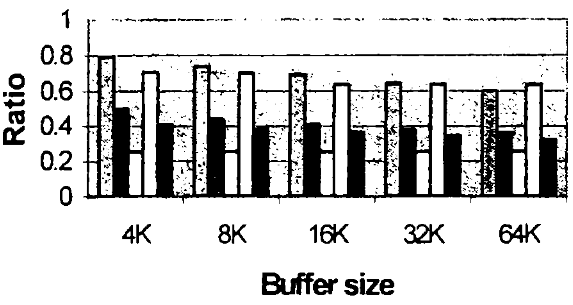

Fig. 1. Compression ratio as function of buffer size

As the buffer length is increased, the compression ratio improves, except for the object code file; still, the buffer should not be too large, as the running time can reach high values.

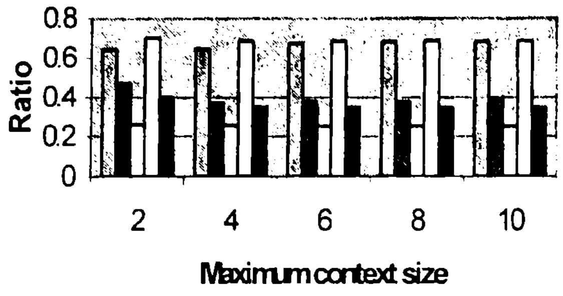

Fig. 2 Compression ratio as function of maximum context size

The maximum context size does not appear to make a lot of difference; it probably helps occasionally to have a larger context, but overall it seems to be less important. The default value of this parameter should be set to 4.

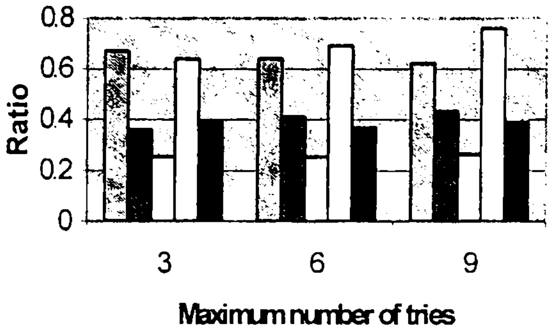

Fig. 3 Compression ratio as function of maximum number of tries

More tries, more errors; if it does not get right fast, chances are it will get wrong in the end. The exception here is represented by the Romanian text, for which more attempts available is good news.

As a conclusion, a maximum context length of 4-6 characters, with a maximum number of tries between 3 and 6, and a buffer as large as possible should yield the best ratios.

V. REMARKS

The present implementation of the symbol ranking text compression method does not completely follow Fenwick's work. The two omitted steps can increase compression and speed. Further optimization is recommended, such as RLE – an implementation by Fenwick [8] uses RLE and hash tables for a fast compressor. Although the compressor is intended to process human languages, it performs good on the object code file.

REFERENCES

[1] Shannon, C. E., "Prediction and Entropy of Printed English", *Bell System Technical Journal*, Vol. 30, pp. 50-64, January 1951

[2] Fenwick, P., "Symbol Ranking Text Compression with Shannon Recordings", *Journal of Universal Computer Science*, Vol. 3, No. 2, pp. 70-85, February 1997

[3] Bentley, J. L., Sleator, D. D., Tarjan R. E., and Wei, V. K., "A Locally Adaptive Data Compression Algorithm", *Communications of the ACM*, Vol. 29, No. 4, pp. 320-330, April 1986

[4] Howard, P. G., and Vitter J. S., "Design and Analysis of Fast Text Compression Based on Quasi-Arithmetic Coding", *Data Compression Conference*, pp. 98-107, IEEE Computer Society, Los Alamitos, California, 1993

[5] Burrows, M., and Wheeler, D. J., "A Block-Sorting Lossless Data Compression Algorithm", SRC Research Report 124, Digital Systems Research Center, Palo Alto, California, May 1994, available at gatekeeper.dec.com/pub/DEC/SRC/research-reports/SRC-124.ps.Z

[6] Bloom, C., "LZP: A New Data Compression Algorithm", *Data Compression Conference*, Vol. 3, No. 2, pp. 70-85, IEEE Computer Society, Los Alamitos, California, 1996.

[7] The Calgary Corpus can be found on the Internet at the address ftp://ftp.cpsc.ucalgary.ca/pub/projects/text.compression.corpus

[8] Fenwick, P. M., "Symbol Ranking Text Compressors: Review and Implementation", *Software Practice and Experience*, Vol. 28, No. 5, pp. 547-559, April 1998