Buletinul Ştiinţific al Universităţii "Politehnica" din Timişoara

Seria ELECTRONICĂ şi TELECOMUNICAŢII
TRANSACTIONS on ELECTRONICS and COMMUNICATIONS

Tom 49(63), Fascicola 2, 2004

Evaluation of Parameters Used in Lossless Text Compression With the Burrows-Wheeler Transform

Radu Rădescu, Ionuţ Bălăşan[1]

Abstract – This paper presents a study of parameters involved in the lossless text compression methods using the Burrows-Wheeler Transform (BWT). BWT (also known as Block Sorting) is one of the most efficient techniques used in data compression. Its purpose is to preprocess the text before applying a compression algorithm, thus providing a better use of the inner redundancy of the text. BTW converts the original blocks of data into a format that is extremely well suited for compression. This paper deals with the choice of the range for the block length for different types of text files, evaluating the compression ratio and compression time.

Keywords: Block Sorting, Move-to-Front (MFT), Run Length Encoding (RLE), Huffman coding, arithmetic coding

I. INTRODUCTION

With the Burrows-Wheeler Transform [1], the compression algorithm is the following. Given a text file, the Burrows-Wheeler Transform is applied on it. This produces a new text, which is suitable for a Move-to-Front encoding [2], [4] (since it has a great number of sequences with identical letters). The result is another new text, which is more suitable for Huffman or arithmetic encoding, usually preceded by a Run-Length Encoding (RLE), [3] since this text produces many small numerical values. The only step that actually performs compression is the third one (the statistical algorithm). The two other steps are meant to ensure that the Huffman/arithmetic encoding [6] is able to compress the data efficiently.

II. BURROWS-WHEELER TRANSFORM

The transform divides the original text into blocks of the same length, each of them being processed separately. The blocks are then rearranged using a sorting algorithm. This is why it is also called *Block Sorting*. [1] The resulting block of text contains the same symbols as the original, but in a different order. Sorting the rows will be the most complex and time consuming task in the algorithm, but present implementations can perform this step with an acceptable complexity. The transformation groups similar symbols, so the probability of finding a character close to another instance of the same character increases substantially. The resulting text can be easily compressed with fast locally adaptive algorithms, such as Move-to-Front coding combined with Huffman or arithmetic coding.

The Block Sorting algorithm transforms the original string S of N characters by forming all possible rotations of those characters (cyclic shifts), followed by a lexicographical sort of all of the resulting strings. The output of the transform is the last character of the strings, in the same order they appear after sorting. All these strings contain the same letters but in a different order. One of them is the original string S. An index of the string S is needed, because its position among the sorted strings has to be known in order to reverse the transform.

The transform is reversible because the output is a string containing the same letters as the input. By performing a lexicographical sorting, a string identical to the first column of the transform matrix M is obtained. Starting only with the last column of the matrix (the transform result), the first column of the matrix is easily recognizable. The unsorting column U requires the use of a *transformation vector* T [5]. The transformation vector gives the correspondence between the characters of the first column and the following ones, found in the last column. The procedure begins with a starting point and thus the rows contained in the original string are identified.

Since BWT groups closely together symbols with a similar context, the output can be more than two times smaller than the output obtained from a regular compression. Compressing a text file with the Burrows-Wheeler Transform can reduce its size while the compression without the transform gave a weaker output. The compression method used in both cases consists of the three stages following BWT: Move-to-Front, Run-Length Encoding and arithmetic coding.

[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

III. MOVE-TO-FRONT

Move-to-Front encoding technique inputs a string and outputs a series of numbers, one for each character in the input string. All the 256 characters in the ASCII code will have correspondents in a list with 256 numbers. It can be made an optimization by putting the most often characters on the first places in the list so that they will be coded with small numbers. If the input string has an important number of sequences containing the same letter in a row, then th utpu r w ... y m... a es. hi means that if an input string has numerous sequences containing the same letter in a row, then the Move-to-Front encoding could be performed first. Huffman or arithmetic encoding could be applied afterwards. For the Huffman coding table an adequate choice is to use less space for small numbers than for greater numbers. The result should be a shorter sequence than the origi..al o...

IV. COMPRESSION PARAMETERS

In order to evaluate th p fi m c s BWT (... compression ratio and the time needed to perform the compression), i is sui able o perform he tests on different types of text files and to vary the block l...g... of the cu...e...l... ...d i...... Tn ... fl... are text files (.txt, .ppt, and .doc) but also a .bmp file, containing a screen shot.

The main goal of the test is to evaluate the contribution of BWT in the overall result of the compression process. The steps of the compression method are the following:

- Run-Length Encoding;
- Burrows-Wheeler Transform;
- Move-To-Front;
- Run-Length Encoding;
- Arithmetic compression.

Initially, the complete 5-step algorithm was performed for the test file set and then the algorithm was performed again, omitting the 2nd step (BWT). Table 1 presents the original dimensions of the test files.

Table 1

File name	*Dimension* (kB)
fis1.doc	85
fis2.doc	694
fis3.doc	858
fis4.ppt	90
fis5.ppt	152
img.bmp	2305

The compression results for both cases are shown in Figure 1 (on the left – with BWT, on the right – without BWT).

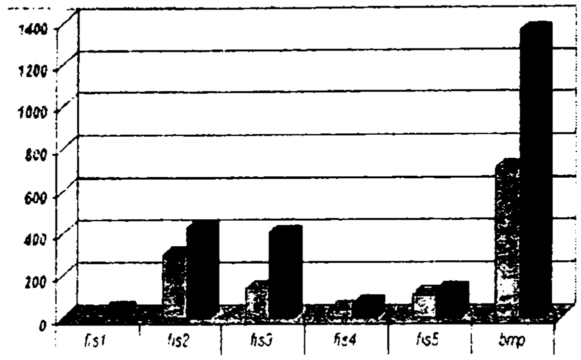

Fig 1 Compression ratios with and without BWT

For both cases (with and without BTW), the compression time was estimated using the same set of test files (in the same order, from left to right). The results are shown in Figure 2.

Fig. 2 Compression time with and without BWT

The obvious conclusion is that BWT improves essentially the compression ratio (two times, in average) with the price of increasing the compression time, but only for certain files from the test set.

In order to estimate the performances of the transform as function of block length (in the compression process), a test file (.doc) of 858 kB was used. The block length represents the information transformed by BWT and compressed at a time (on a processing stage). The number of stages performed to obtain the compressed file is calculated as the overall dimension of the file divided by the block dimension. Taking into account that the file is read binary and the output is stored on bytes (characters of 8 bits) it results a number of 858,000 symbols for the test file. It is recommended to choose the block length sufficiently large in order to exploit the redundancy within. The compression results for different values of block length are presented in Table 2.

The dimension of the compressed file is constantly decreasing until the block length exceeds 320,000 symbols. Beyond this value, it appears a limitation and then a slight increasing of the resulting archive.

To represent the corresponding graphic a logarithmic scale was used in order to largely emphasize the range of the values for the block length. The dimension of the compressed file as function of block length is shown in Figure 3.

Table 2

Block length ($\times 10^3$)	*Compressed file* (kB)
0.1	671
1	256
2	213
3	196
5	180
10	165
50	145
100	142
250	139
286	134
300	132
325	131
350	132
400	134
600	138
800	138

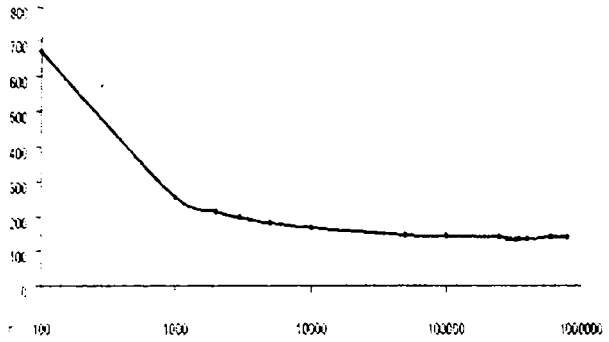

Fig. 3. Compressed file dimension as function of block length

The dimension of the compressed file is constantly decreasing with block length. For high values of the block length, the compression ratio (calculated as original file dimension divided by compressed file dimension) is stabilized to the value of 6.35. Generally, the block length could be about 200,000 bytes. In this case, both compression ratio and compression time reach their optimal values.

In order to evaluate the algorithm complexity once the BWT was introduced, the compression time for both cases is compared. The BWT implies the permutation of the symbols within the file, as well as the sorting of the permutations.

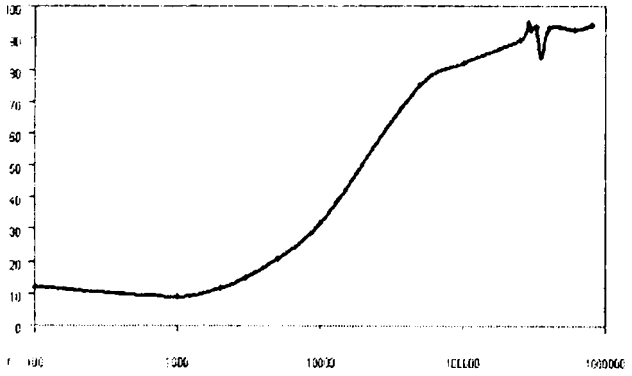

Fig. 4. Compression time as function of block length

Therefore, it is obvious that the required time will increase, depending on the dimension of the block of processed data.

The compression time as function of block length is shown in Figure 4. For large values of the block length, the compression time substantially increases in the case of applying the BWT. This result could be explained not only by the presence of the BWT but also by the adaptive arithmetic compression, which supposes a two-stage processing of the block and an adjustment of the codewords, depending on their frequencies and, eventually, on the number of symbols.

V. COMPARISON WITH STANDARD COMPRESSORS

WinRAR, WinAce and WinZip were chosen among the usual compression programs, in order to compare the performances of the algorithm presented above, applied on the same 5-file test set. The dimensions of the compressed files (using the 3 standard compressors and the BWT algorithm) are shown in Figure 5, for the considered test files.

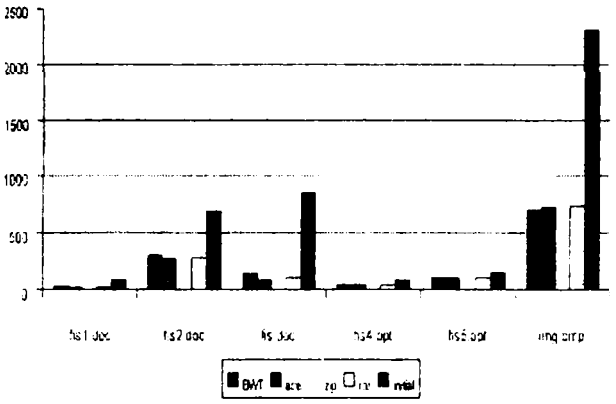

Fig. 5. Dimension as function of block length for BWT, WinAce, WinZip, WinRAR compressed files and original files

The advantages of using the BTW algorithm for all files (and especially for the image file) are obvious. The compression ratio of the BWT algorithm is very close to the average of the compression ratios of the standard compressors.

V. CONCLUSIONS AND REMARKS

The compression technique based on BWT provides good results in comparison with the general-purpose compressors. The algorithm has a high degree of generality and could be applied on the majority of file types (text, image or other files).

BWT uses the sorting of symbols in the original file, and the data processing is performed on blocks obtained by d...d...g .h. s.urc. fil.. A.. .mp.....
issue in optimizing the performances of the BWT algorithm is to choose an adequate value of the block

length. From this point of view, one have to take into account both the compression performances and the required computing resources. To get a good compression ratio and an acceptable compression time, the block length could be situated around 200,000 bytes. For block length less than 100,000 bytes, the compression ratio is sensibly decreased, as well as the compression time. An excessive increasing of block length (over 800,000 bytes) produces an unacceptable compression time.

Generally, one can observe a constancy of the compression ratio for the recommended value of the block length (200 kB), due to the high redundancy within the text or the image file. This value could be considered an upper bound for the block length, in order to assure a satisfactory result.

The BWT algorithm can be applied on any type of data because the inverse transform is performed with no losses of information. Hence, BTW modifies the symbol positions but it does not change the probability distribution. As a result, the complexity of an implementation of the overall compression method (including the other four steps) does not exceed the similar values of the classic lossless compression standards, based on LZW-type algorithms.

The BWT represents an efficient data processing method that could be successfully integrated in any compression technique for general purpose.

REFERENCES

[1] M. Burrows and D. J. Wheeler, "A Block-Sorting Lossless Data Compression Algorithm", 1994, report available at: http://gatekeeper.dec.com/pub/DEC/SRC/research-reports/abstracts/src-rr-124.html.

[2] M. Nelson, "Data Compression with the Burrows-Wheeler Transform", September 1996, available at: http://dogma.net/markn/articles/bwt/bwt.htm.

[3] M. A. Maniscalco, "A Run Length Encoding Scheme for Block Sort Transformed Data", 2000, available at: http://www.geocities.com/m99datacompression/papers/rle/rle.html.

[4] P. M. Fenwick, "Block Sorting Text Compression", 1996, available at: ftp.cs.auckland.ac.nz.

[5] T. C. Tell, J. G. Cleary and I. H. Witten, *Text Compression*, Prentice Hall, Englewood Cliffs, NJ, 1990.

[6] R. Rădescu, *Compresia fără pierderi: metode şi aplicaţii*, Matrix Rom, Bucharest, 2003.