ROMANIAN JOURNAL OF INFORMATION SCIENCE AND TECHNOLOGY
Volume **12**, Number 1, 2009, 101–115

Transform Methods Used in Lossless Compression of Text Files

Radu RĂDESCU

Polytechnica University of Bucharest,
Faculty of Electronics, Telecommunications and Information Technology
Applied Electronics and Information Engineering Department
1–3, Iuliu Maniu Blvd, sector 6, Bucharest, Romania

E-mail: `rradescu@atm.neuro.pub.ro`

Abstract. This paper presents a study of transform methods used in lossless text compression in order to preprocess the text by exploiting the inner redundancy of the source file. The transform methods are Burrows-Wheeler Transform (BWT, also known as Block Sorting), Star Transform and Length-Index Preserving Transform (LIPT). BWT converts the original blocks of data into a format that is extremely well suited for compression. The chosen range of the block length for different text file types is presented, evaluating the compression ratio and compression time. Star Transform and LIPT applied to text emphasize their positive effects on a set of test files picked up from the classical corpora of both English and Romanian texts. Experimental results and comparisons with universal lossless compressors were performed, and set of interesting conclusions and recommendations are driven on their basis.

1. Introduction

One of the new approaches in lossless text compression is to apply a reversible lossless transformation to a source file before applying any other existing compression algorithm. The transformation is meant to make the file compression easier [1]. The original text is offered to the transformation input and its output is the transformed text, further applied to an existing compression algorithm. Decompression uses the same methods in the reverse order: decompression of the transformed text first and the inverse transform after that.

Using 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], [14], [15] is able to compress the data efficiently.

The experimental results [3] measuring the performances of the preprocessing methods are given using three reference test corpora: Calgary, Canterbury, and Guttenberg, but also a set of representative text files for the Romanian language.

2. The Burrows-Wheeler Transform

Compression implementation using the Burrows-Wheeler Transform consists of *three stages* which are described below (see Fig. 1):

1. The *Block Sorting algorithm*, which reorders the data making it more compressible, with clusters of similar symbols and many symbol runs;
2. The permuted symbols are recoded by a *Move-To-Front* (MTF) re-coder;
3. Compression using a *statistical compressor* such as Huffman or arithmetic, usually preceded by run-length encoding (RLE).

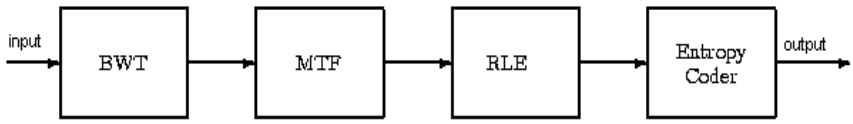

Fig. 1. The block diagram of the Burrows-Wheeler Transform (BWT).

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.

3. The Star (*) Transform

The main idea for the Star Transform [10] is to define a unique signature for each word replacing the letters of the word by a special character (*) and to use a minimum number of characters in order to identify precisely the specified word. For the English dictionary (D) with the size of 60 000 words, it was observed that at most two characters from the original word are needed in order to preserve its unique identity. In fact, there is no need to keep a letter from the original word as long as a unique representation can be defined.

The dictionary is divided into D_s sub-dictionaries containing words with length $1 < s < 22$, because the maximum length of an English word is 22 letters and because there are two one-letter words („a" and „I").

For the words in the D_s dictionary the following 6-step coding scheme is used:

Step 1 The first word is represented as a sequence of s characters *.

Step 2 The next 52 (= 2×26) words are represented by a sequence of (s–1) characters * followed by a single alphabet letter (a, b, ..., z, A, B, ..., Z).

Step 3 The next 52 words have a similar encoding except that the letter is placed in the sequence on the second position from the right.

Step 4 The procedure continues until every letter was placed in the sequence on the first position from the left.

Step 5 The next group of words has (s–1) characters * and the 2 remaining positions are occupied by unique pairs formed by alphabet letters.

Step 6 The procedure continues in order to obtain a total of 53^s unique codes (see Fig. 2).

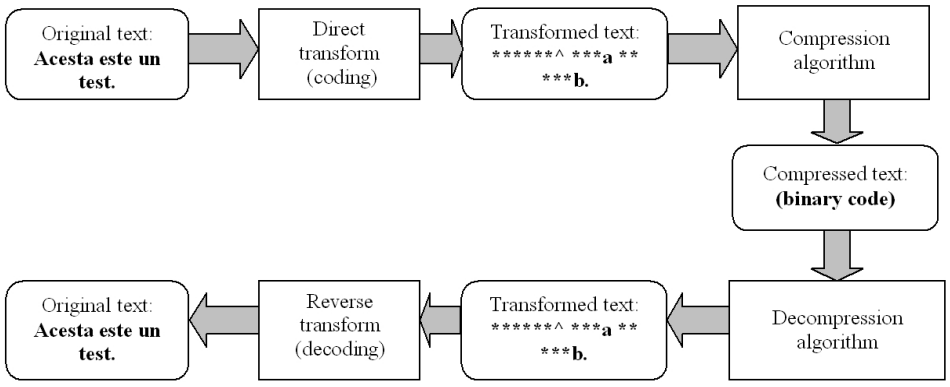

Fig. 2. The text compression model using a reversible lossless transformation.

Most part of these combinations is never used. For instance, for $s = 2$ there are only 17 words in English and for $s = 8$ there are about 9 000 words in the English dictionary.

The transformed text can be processed by using any lossless text compression algorithm, including the Bzip2 algorithm, where the text undergoes two transformations: the Star Transform and the Burrows-Wheeler Transform (BWT) [1].

4. The LIPT Transform

It was noticed that the majority of words in the English language have a length ranging between 1 and 10 letters. Most of them have a length ranging between 2 and 5 letters. The length and the frequency of words were a solid base on creating the LIPT transform method [12], [17]. This can be considered a first phase of a compression algorithm with many stages, like the Bzip2, that includes the RLE encoding [6], the BWT transform [1] method and the MTF [1] and Huffman encoding [6], [14], [15]. LIPT can be used as an additional component in the Bzip2 algorithm, before the RLE encoding, or it can also replace it.

In addition to the Star Transform [10], some changes were made to increase the speed performance of the LIPT transform method. At the Star encoding, the search of a certain word in the coding phase and repeating it in the decoding phase takes time, this causing the increase of the execution time. This situation can be improved with an initial sort out in lexicographic order of the words from the dictionary, then a binary search in the sorted dictionary in the encoding and decoding phase.

Another new idea that was applied is that in the decoding phase words can be accessed in a random order, to produce a rapid decoding. Generating the addresses of the words in the dictionary not in a numeric format, but using the alphabet letters accomplishes this. For an address, a maximum of three letters is used, and these letters increase the redundancy that can be exploited by the compression algorithm.

LIPT uses a dictionary of the English language composed by 59 951 words, with a size of approximately 0.5 MB. The transformation dictionary has approximately 0.3 MB. Between the transformation dictionary and the English, one is a one to one correspondence. The words that are not found in the dictionary are left unchanged in the coded text.

To create the LIPT dictionary, the English dictionary needs to be sorted depending on the word length, and every block of a specific length to be sorted in descending order of the frequency of the words.

The encoding and decoding processes can be put together. We assume that both the compressor and the decompressor have access to the same D dictionary and its D_{LIPT} correspondent.

The encoding steps are:

1. The words from the input text are searched in the D dictionary.

2. If the input text is found in the D dictionary, then the position and the number of its length block are marked down and the adequate transformation is searched at the same position and same length block in the D_{LIPT} dictionary. This transformation is the encoding of the input word. It this is not found in the D dictionary that it is transferred unchanged.

3. Once the input text is transformed according to step 1 and 2, this is sent to a compressor (e.g., Bzip2, PPM [13], [16], [18], [19] etc.)

The decoding steps are:

1. The compressed received text is first decompressed using the same compressor used in the coding phase the result being the LIPT transformed text.

2. To the decompressed text, an inverse transformation is applied. The words preceded by the "*" character are the transformed ones, and those without it are unchanged, so they do not need the inverse transformation. The transformed word is then interpreted like this: the length character indicates the length block where the decoded word is found, the next three symbols indicate the *offset* where the word is situated in that block and there can also be a capitalization mask. The word is then searched in the D dictionary according to the above parameters. The transformed words are then replaced with the corresponding ones from the D dictionary.

3. The capitalization mask is applied.

5. Experimental Results

A. In order to evaluate the performances of BWT (the compression ratio and the time needed to perform the compression), it is suitable to perform the tests on different types of text files and to vary the block length of the currently processed input. The test files 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 2^{nd} step (BWT). The compression results for both cases are shown in Fig. 3 (on the left – with BWT, on the right – without BWT).

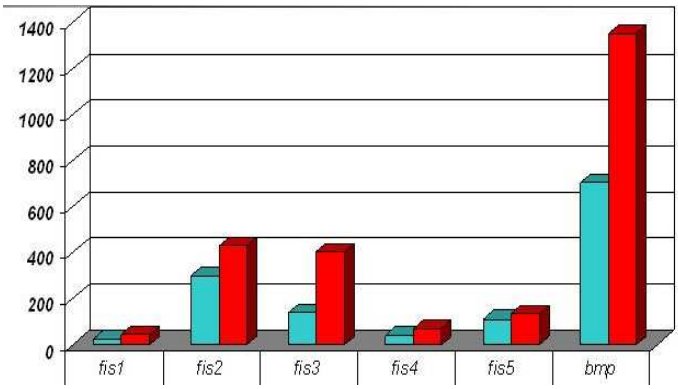

Fig. 3. Compression ratio (%) with and without BWT for different files.

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 Fig. 4.

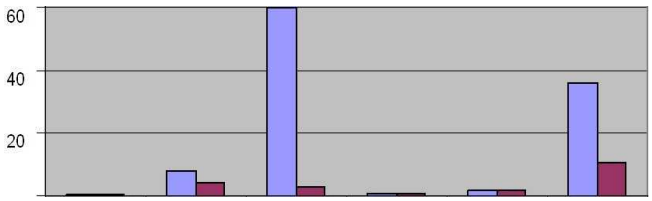

Fig. 4. Compression time (ms) with and without BWT for different files.

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 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 Fig. 5.

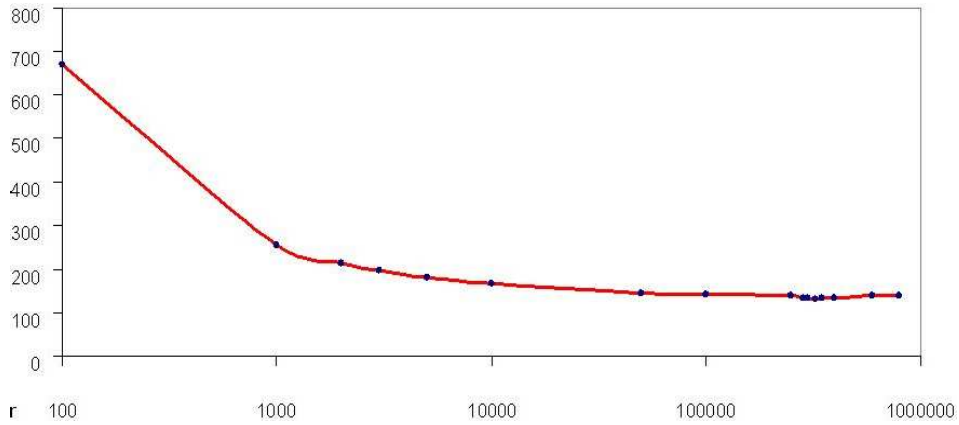

Fig. 5. Compressed file dimension (kB) as function of block length (number of characters).

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

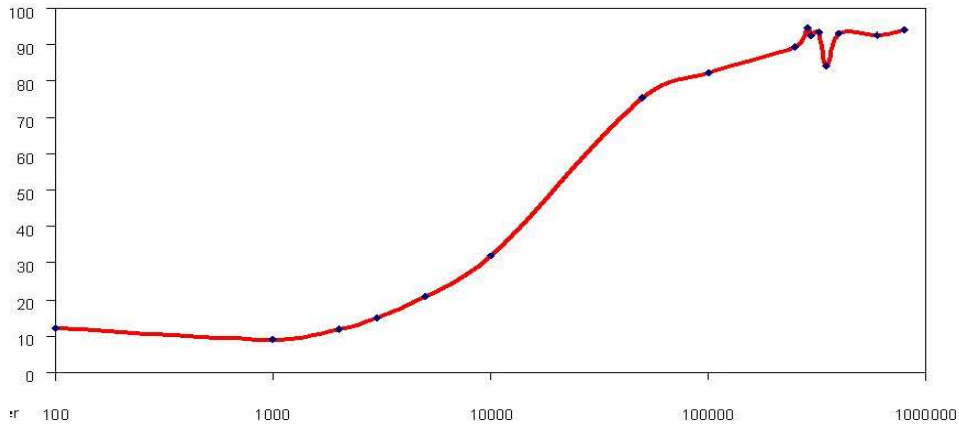

Fig. 6. Compression time (ms) as function of block length (number of characters).

The compression time as function of block length is shown in Fig. 6. 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.

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 Fig. 7, for the considered test files.

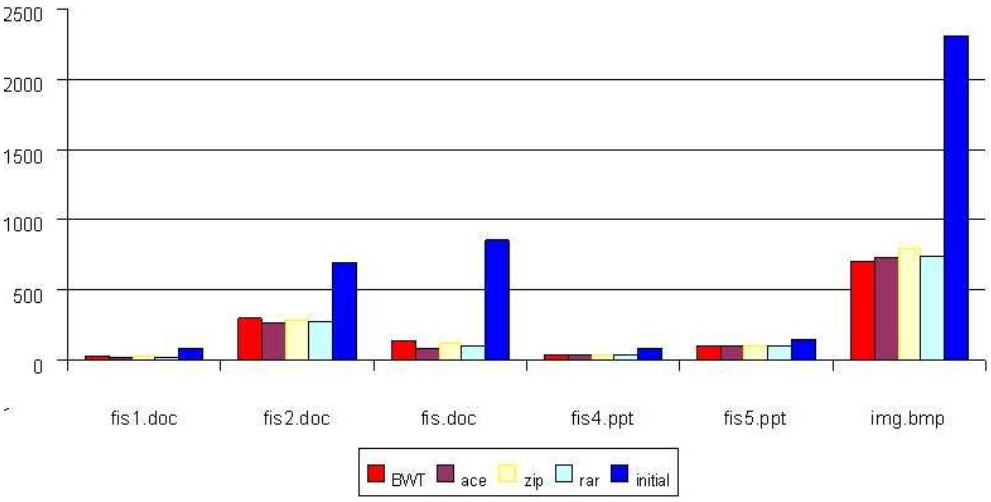

Fig. 7. Dimension (kB) as function of block length for BWT, WinAce, WinZip, WinRAR compressed files and original files.

The advantages of using BTW 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.

B. In order to test the influence of the Star Transform on the compression ratio we have performed a set of experiments using text files with different size and content. The files (book1 – 750.74 kB, book2 – 596.54 kB, paper1 – 51.91 kB, paper2 – 80.27 kB) are part of the Calgary Corpus [11].

The compression ratios obtained for the original and the transformed (*) files are presented in Figs. 8–11. Several universal compressors were used. WinRar, WinZip, and WinAce were chosen among the usual compression programs in order to compress the set of text files. Another Huffman-based compressor was used in the tests [7]–[9]. A usual English dictionary was involved.

The usual compressors show something between a small improvement and a small deterioration of the compression ratio (due to the internal compression mechanisms, already adapted and optimized for text compression), but the Huffman-based compressor emphasizes an obvious progress using the Star Transform.

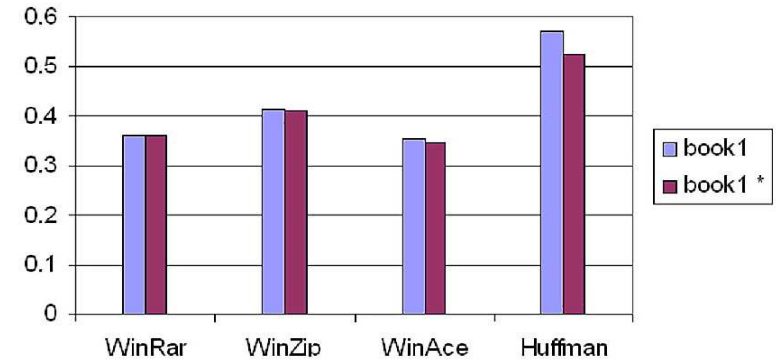

Fig. 8. Compression ratios obtained for book1, original (left) and coded with * (right).

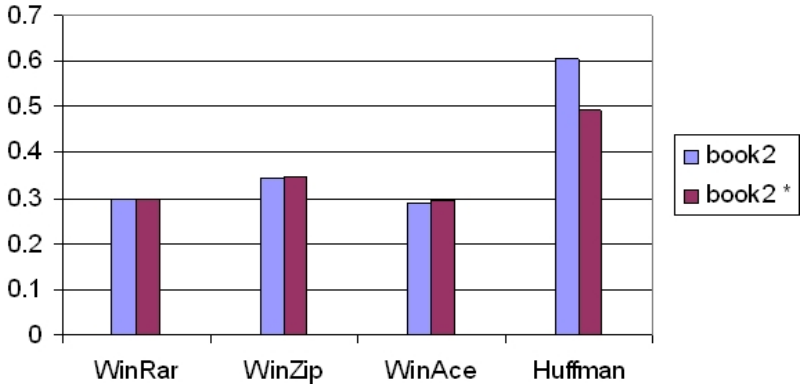

Fig. 9. Compression ratios obtained for book2, original (left) and coded with * (right).

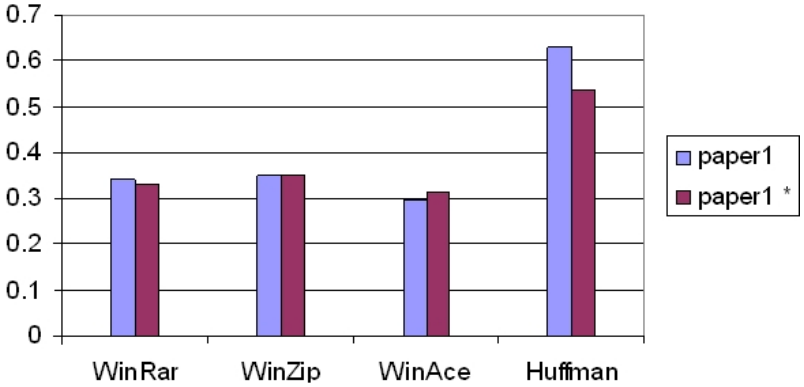

Fig. 10. Compression ratios obtained for paper1, original (left) and coded with * (right).

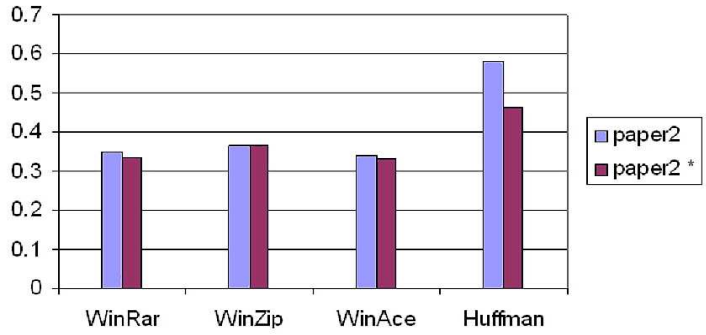

Fig. 11. Compression ratios obtained for paper2, original (left) and coded with * (right).

In order to test the compression on Romanian text (with diacritics) an extract of the Penal Code of Romania was chosen as test file (344 kB). The tests were performed using the same compressors as for the English text files (WinRar, WinZip, WinAce, and Huffman). The code dictionary was constructed based on several Romanian texts. Figure 12 shows the subsequent results.

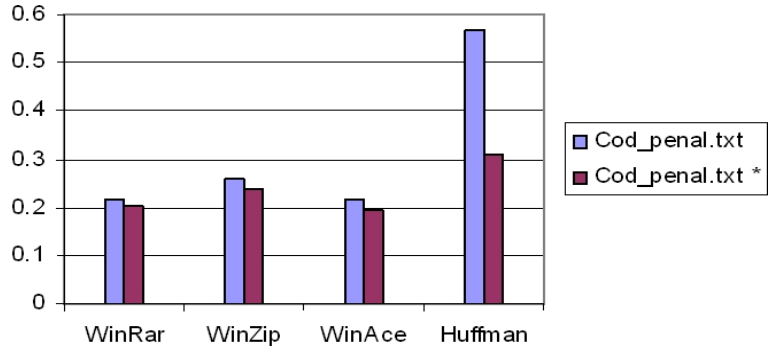

Fig. 12. Compression ratios obtained for Cod_penal.txt, original (left) and coded with * (right).

This time it is obvious that in every case a clear improvement was obtained, with a more spectacular value for the Huffman-based compressor.

C. The performance of the LIPT transform was measured using the following compression algorithms: Bzip2, PPMD (level 5) [13] and Gzip. The results were compared depending on the mean compression ratio, measured in BPC (bits per character). The mean compression ratio is the average of all the compression ratios from the test frame.

Combining the results on all test sets and computing the mean compression ratio on all text files, the results can be put together like this:

1. The mean compression ratio using only the Bzip2 algorithm is 2.28 and using the Bzip2 algorithm along with the LIPT transform is 2.16, emphasizing a 5.24% improvement.

2. The mean compression ratio using only the PPMD algorithm (level 5) is 2.14 and using the PPMD (level 5) algorithm together with the LIPT transform is 2.04, emphasizing a 4.46% improvement.

3. The mean compression ratio using only the Gzip algorithm is 2.71 and using the Gzip together with the LIPT transform is 2.52, emphasizing a 6.78% improvement.

The performance obtained in compression with the use of the LIPT transform has a disadvantage, which is the augmentation of the rendering time. For *off-line* archiving applications, this disadvantage could be neglected if it can be obtained a substantial gain in disk space. The bigger times that appear at the archiving and extracting are due to the frequent access of the original dictionary and the dictionary with codes. To face these problems there have been developed efficient data structures that augment the dictionary access speed and management techniques that use caching methods.

To test the influence of the LIPT transform on the compression ratio some experiments on files with different lengths and contents were performed. The used files are part of the Calgary Corpus collection [11].

Figures 13–16 show a comparisons between the compression ratios obtained for original files and coded files using the LIPT transform.

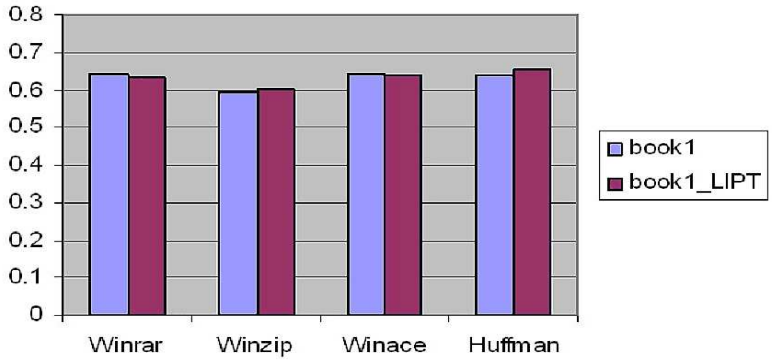

Fig. 13. Compression ratios of the archivers for book1 original (left) and with LIPT (right).

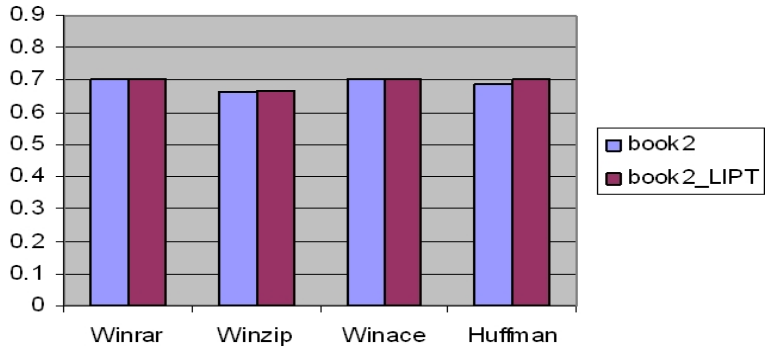

Fig. 14. Compression ratios of the archivers for book2 original (left) and with LIPT (right).

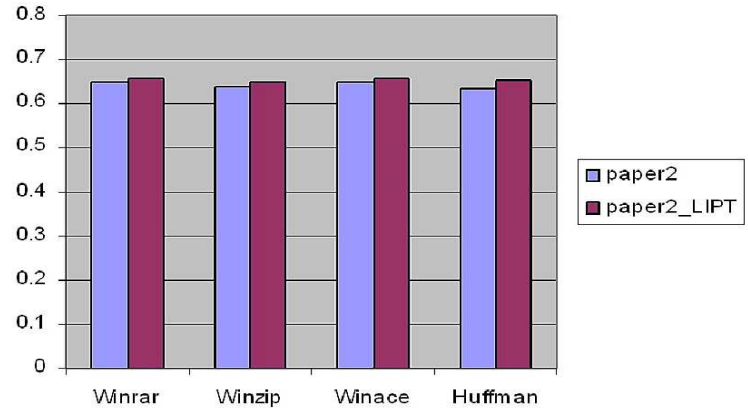

Fig. 15. Compression ratios of the archivers for paper1 original (left) and with LIPT (right).

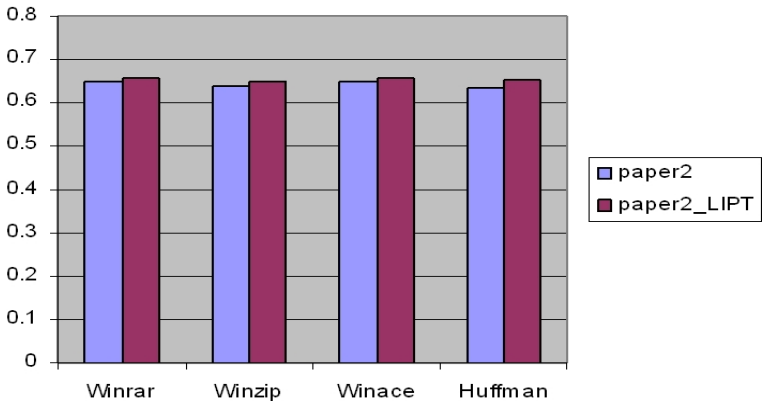

Fig. 16. Compression ratios of the archivers for paper2 original (left) and with LIPT (right).

For compression there have been used many applications. From the usual compression applications, WinRar, WinZip and WinAce have been chosen for the set of files. For testing, a Huffman coding compressor was used. For WinRar, the last version was chosen, 3.70. WinZip uses the version 11.1 and WinAce uses the version 2.5. As a dictionary, a common English one was used. The compression ratio was calculated using the formula: $R = 1 - A/B$, where R = the compression rate, A = number of bits after the compression, and B = number of bits after the compression.

In order to test the application for texts in the Romanian language (with diacritics), it was chosen the "Regulament de ordine interioar" input text. The Regulament_ordine.txt file has a size of 84 792 B. The tests were performed using the same compressors as for the English language files (WinRar, WinZip, WinAce, and Huffman). The dictionary of codes was made up based on Romanian language texts. The compression results for the Romanian text file are presented in Fig. 17.

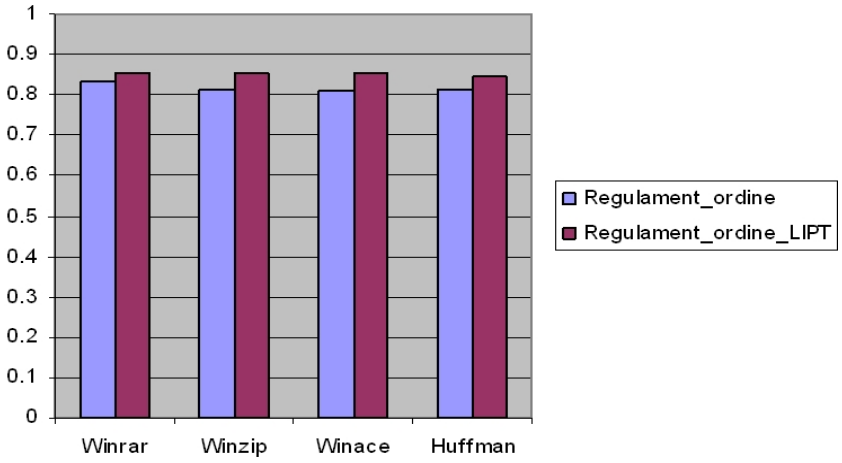

Fig. 17. Compression ratios of the archivers for Regulament_ordine original (left) and with LIPT (right).

6. 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 dividing the source file. An important 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 has 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.

The text files used for Star Transform and LIPT are either extracted from the Calgary Corpus or specially conceived (in the case of Romanian text). The size of the compressed files was compared with the original files size. The performed tests with and without Star Transform and LIPT pointed out an important increasing in compression performance (especially) in the case of the Huffman-based compressor.

The Huffman compression encodes the most frequent character (*) with a one-bit codeword. Improvements of up to 33% for Star Transform and up to 7% for LIPT were obtained for the Huffman algorithm. As an average result, Star Transform and LIPT prove their utility as lossless text compression preprocessing methods.

References

[1] BURROWS M., WHEELER D. J., *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] NELSON M., *Data Compression with the Burrows-Wheeler Transform*, September 1996, available at: `http://dogma.net/markn/articles/bwt/bwt.htm`.

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

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

[5] TELL T. C., CLEARY J. G., WITTEN I. H., *Text Compression*, Prentice Hall, Englewood Cliffs, NJ, 1990.

[6] RĂDESCU R., *Compresia fără pierderi: metode şi aplicaţii*, Matrix Rom, Bucharest, 2003 (in Romanian).

[7] FRANCESCHINI R., KRUSE H., ZHANG N., IQBAL R., MUKHERJEE A., *Lossless, Reversible Transformations that Improve Text Compression Ratios*, *Preprint of the M5 Lab*, University of Central Florida, 2000.

[8] FRANCESCHINI R., MUKHERJEE A., *Data compression using text encryption*, *Proceedings of the Third Forum on Research and Technology, Advances on Digital Libraries*, ADL, 1996.

[9] KRUSE H., MUKHERJEE A., *Data compression using text encryption*, *Proceedings of Data Compression Conference*, IEEE Comput. Soc., Los Alamitos, CA, 1997.

[10] NELSON M. R., *Star Encoding*, Dr. Dobb's Journal, August, 2002.

[11] *Calgary Corpus*: `ftp://ftp.cpsc.ucalgary.ca/pub/projects/text.compression.corpus`.

[12] AWAN F. S., ZHANG N., MOTGI N., IQBAL R. T., MUKHERJEE A., *LIPT: A Reversible Lossless Text Transform to Improve Compression Performance*, *Proceedings of Data Compression Conference*, Snowbird, UT, 2001.

[13] RĂDESCU R., HARBATOVSCHI C., *Compression Methods Using Prediction By Partial Matching*, *Proceedings of the 6th International Conference Communications 2006*, pp. 65–68, Bucharest, Romania, 8–10 June 2006.

[14] RĂDESCU R., NICULESCU I., *Rezultate recente în compresia fără pierderi folosind metode Huffman adaptive cu implementări eficiente*, Revista EEA Electronică, Electrotehnică, Automatică, Editura ICPE, Bucureşti, **vol. 54**, no. 4, oct.–dec. 2006, pp. 1–4, ISSN 1582-5175 (in Romanian).

[15] RĂDESCU R., LICULESCU G., *Efficient Implementation of Adaptive Huffman Methods in Lossless Compression*, *Proceedings of the Fifth International Workshop on Optimal Codes and Related Topics OC2007*, Balchik, Bulgaria, 16–22 June, 2007, pp. 209–215, ISSN 1313-1117.

[16] RĂDESCU R., LICULESCU G., *Table Look-Up Lossless Compression Using Index Archiving*, *Proceedings of the Fifth International Workshop on Optimal Codes and Related Topics OC2007*, Balchik, Bulgaria, 16–22 June 2007, pp. 216–221, ISSN 1313-1117.

[17] RĂDESCU R., *Lossless Text Compression Using the LIPT Transform*, *Proceedings of the 7th International Conference Communications 2008 (COMM2008)*, pp. 59–62, Bucharest, Romania, 5–7 June 2008, ISBN 978-606-521-008-0.

[18] RĂDESCU R., *Lossless Compression Tool for Medical Imaging*, *Proceedings of the 6th International Symposium on Advanced Topics in Electrical Engineering, ELTH & AIEER*, pp. 265–267, November 20–21, 2008, Bucharest, Romania, Printech Press, ISBN 978-606-521-137-7.

[19] RĂDESCU R., BONTAŞ C., *Design and Implementation of a Dictionary-Based Archiver*, Sci. Bull., Electrical Engineering Series C, University Politehnica of Bucharest, **vol. 70**, no. 3, pp. 21–28, 2008, ISSN 1454-234X.

[20] `http://gatekeeper.dec.com/pub/DEC/SRC/research-reports/abstracts/src-rr-124.html`

[21] `http://www.dogma.net/markn/articles/Star/`

[22] `http://www.codeguru.com/forum/`

[23] `http://www.codersource.net/codersource_mfc_prog.html`

[24] `http://www.winace.com`

[25] `http://www.winrar.com`

[26] `http://www.winzip.com`

[27] `http://www.y0da.cjb.net`