JBE. CONCEPT AND APPLICATION

Lydia Mitkovets, Daniel Sidorov, students, Alevtina Gourinovitch,

PhD of Belarusian State University of Informatics and Radioelectronics, Minsk, 6 P.Brovki str.

Annotation. To save the information inside storage users try to reduce the files size to minimum by using data compression software. It is a new algorithm for data compression in this article. It is j-bit encoding (JBE). This algorithm manipulates each bit of data inside file to minimize the size without losing any data after decoding. It is classified lossless compression. This basic algorithm is combining with other data compression algorithms to optimize the compression ratio. The implementation of this algorithm consists in a combination of various data compression algorithms.

Keywords: data packaging, compression, compression encoding, source encoding.

Introduction

Data compression is an algorithmic transformation of data to reduce the amount of data it occupies. This algorithm is applied for efficient usage of storage and data transfer devices.

Compression is the eliminating the redundancy contained in the source data. The simplest example of redundancy is the repetition of fragments in the text (for example, words of natural or machine language). Such redundancy is usually eliminated by replacing the repeated sequence with a reference to the already encoded fragment with an indication of its length. Another type of redundancy is the fact that some values in the compressed data are more common than other ones. The reduction in data volume is achieved by replacing frequently occurring data with short the code word, and rare data with long ones (entropy coding). Compression of data that does not have the property of redundancy (for example, random signal or white noise, encrypted messages) is fundamentally impossible without loss.

At the heart of any compression method is the data source model, or more precisely, the redundancy model. In other words, data compression uses some a priori information about what kind of data has compressed. Without such information about the source, it is impossible to make any assumption about the transformation that would reduce the volume of the message. The redundancy model can be static, immutable for the entire compressed message, or constructed or parameterized at the compression (and recovery) stage.

All data compression methods are differed to two main classes:

- Lossless compression
- Loss compression

When using lossless compression, it is possible to completely restore the original data, loss compression allows you to restore data with distortions that are usually insignificant from the point of view of further use of the restored data. Lossless compression is usually used to the transmission and storage of text data, computer programs, less often-to reduce the volume of audio and video data, digital photos, etc., in cases where distortion is unacceptable or undesirable. Loss compression is significantly more efficient than lossless compression. Loss compression is usually used to reduce the volume of audio and video data and digital photos in cases where such reduction is a priority, and full compliance of the original and restored data is not required.

Data compression is a way to reduce storage cost by eliminating redundancies that happen in most files. There are two types of compression, loss and lossless. Loss compression reduced file size by eliminating some unneeded data that won't be recognize by human after decoding, this often used by video and audio compression. Lossless compression on the other hand, manipulates each bit of data inside file to minimize the size without losing any data after decoding. This is important because if file lost even a single bit after decoding, that mean the file is corrupted.

Most compression methods are physical and logical. They are physical because look only at the bits in the input stream and ignore the meaning of the contents in the input. Such a method translates one-bit stream into another, shorter, one. The only way to understand and decode of the output stream is by knowing how it was encoded. They are logical because look only at individual contents in the source stream and replace common contents with short codes. Logical compression method is useful and effective (achieve best compression ratio) on certain types of data [2].

Related Algoritms

A. Run Length Encoding

Run-length encoding (RLE) is one of basic technique for data compression. The idea behind this approach is this: If a data item *d* occurs *n* consecutive times in the input stream, replace the n occurrences with the single pair *nd* [2]. RLE is to compress runs of the same byte. This approach is useful when repetition often occurs inside data. That is why RLE is one good choice to compress a bitmap image especially the low bit one (example 8-bit bitmap image).

B. Burrows-wheeler transform

Burrows-wheeler transform (BWT) works in block mode while others mostly work in streaming mode. This algorithm classified into transformation algorithm because the main idea is to rearrange (by adding and sorting) and concentrate symbols. These concentrate symbols are used for another algorithm to achieve good compression ratios. Since the BWT operates on data in memory, it may encounter files too big to process in one fell swoop. In these cases, the file has to split up and processed a block at a time [3]. To speed up the sorting process, it is possible to do parallel sorting or using larger block of input if more memory available.

C. Move to front transform

Move to front transform (MTF) is another basic technique for data compression. MTF is a transformation algorithm to do not compress data but can help to reduce redundancy sometimes [5]. The main idea is to move to front the symbols that mostly occur, so those symbols will have smaller output number. This technique is to implement optimization for another algorithm likes Burrows-wheeler transform.

D. Arithmetic coding

Arithmetic coding (ARI) is using statistical method to compress data. The method starts with a certain interval, it reads the input file symbol by symbol, and uses the probability of each symbol to narrow the interval. Specifying a narrower interval requires more bits, so the number constructed by the algorithm grows continuously. To achieve compression, the algorithm is the following: a high-probability symbol narrows the interval less than a low-probability symbol, with the result that high-probability symbols contribute fewer bits to the output. Arithmetic coding, is entropy coder widely used, the only problem is its speed, but compression tends to be better than Huffman (other statistical method algorithm) can achieve [2]. This technique is useful for final sequence of the data compression by the combination algorithm and gives the most for compression ratio.

Modified Algorithm

J-bit encoding (JBE) [8] works by manipulate bits of data to reduce the size and optimize input for another algorithm. The main idea of this algorithm is to split the input data into two data where the first data will contain original nonzero byte and the second data will contain bit value explaining position of nonzero and zero bytes. Both data then can be compress separately with other data compression algorithm to achieve maximum compression ratio. The compression process can be describe as following:

1. Read input per byte, can be all types of file.
2. Determine read byte as nonzero or zero byte.
3. Write nonzero byte into data I and write bit ‘1’ into temporary byte data, or only write bit ‘0’ into temporary byte data for zero input byte.
4. Repeat step 1-3 until temporary byte data filled with 8 bits of data.
5. If temporary byte data filled with 8 bits then write the byte value of temporary byte data into data II.
6. Clear temporary byte data.
7. Repeat step 1-6 until end of file is reach.
8. To write combined output data:
   а) Write combined output data;
   б) Write data I.
   в) Write data II.
9. If followed by another compression algorithm, data I and data II can be compress separately before combined (optional).

Figure 1 shows visual compression process for J-bit encoding step-by-step. The inserted original input length is used to information for data I and data II size into the output beginning.

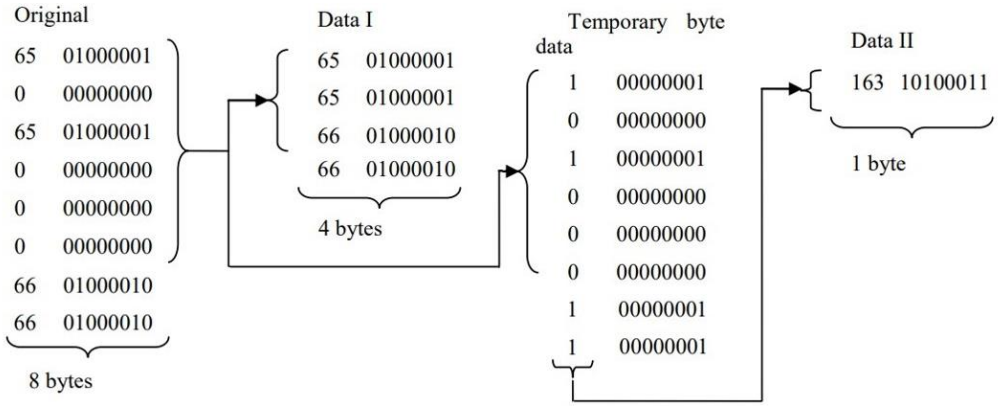

Figure 1. Step by step compression process for J-bit encoding

The decompression process can be describe as following:

1. Read original input length.
2. If was compressed separately, decompress data I and data II (optional).
3. Read data II per bit.
4. Determine whether read bit is ‘0’ or ‘1’.
5. Write to output, if read bit is '1' then read and write data I to output, if read bit is '0' then write zero byte to output.
6. Repeat step 2-5 until original input length is reach.

Variant Combination

Four combinations of data compression algorithm have used to find out which combination with the best compression ratio.

The combinations are:

1. BWT+MTF+ARI.
2. BWT+RLE+ARE.
3. RLE+BWT+MTF+RLE+ARI (as used in [3]).
4. RLE+BWT+MTF+JBE+ARI.

Those combinations have tested with six types of files. Each type consists of 80 samples. Each sample has different size to show real file system condition. All samples are uncompressed, this include raw bitmap images and raw audio without loss compression.

No	Name	Qty	Type	Spec.
1	Image	80	Bitmap Image	Raw 8 bit
2	Image	80	Bitmap Image	Raw 24 bit
3	Text	80	Text Document	
4	Binary	80	Executable, library	
5	Audio	80	Wave Audio	Raw
6	Video	80	Windows Media Video	VBR

Figure 2. Experiment Samples

JBE Application

The structural data compression system looks like this:

Source Data –> Encoder –> Compressed Data –> Decoder –> Recovered Data

In this scheme: the data generated by the source is the source data, and their compact representation is the compressed data. The data compression system consists of an encoder and a source decoder. The encoder converts the source data to the compressed data, and the decoder is to recover the source data from the compressed data. The recovered data generated by the decoder can either exactly match the original data of the source, or slightly differ from them.

In lossless compression systems, the decoder recovers the source data absolutely accurately, so the structure of the compression system is as follows:

Data Vector X –> Encoder –> B (X) –> Decoder - > X

The vector of source data X to be compressed is a sequence X = $(x_1, x_2, \dots, x_n)$ of finite length. The sample x_i (the components of the vector X) has selected from the finite alphabet of data A. In this case, the size of the data vector n is limited, but it can be arbitrarily large. Thus, the source at its output forms as data X a sequence of length n from the alphabet A.

The vector of source data X to be compressed is a sequence *B(X)* = $(b_1, b_2, \dots, b_n)$, размер которой *k* зависит от *X*. It calls *B(X)* the code word assigned to vector X by the encoder (or the code word into which vector X has transformed by the encoder). Since the compression system is non-destructive, the same vector $X_l = X_m$ has to correspond to the same code words *B*(X_l)= *B*(X_m).

Implement BWT+MTF+ARI

Let the input string be "ABACABAA".

1. BWT.

The conversion has implemented by three stages. At the first stage, a table of all cyclic shifts of the input string has compiled. At the second stage, lexicographic (in alphabetical order) sorting of the table rows is performed. In the third step, the last column of the conversion table has selected in the output row. The following example illustrates the described algorithm:

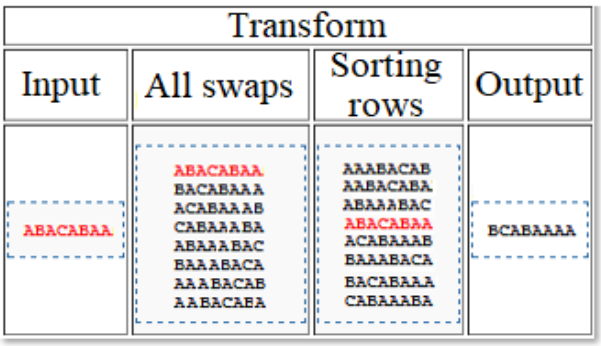

Figure 3. BWT conversion algorithm

Thus, the result of the BWT(s) algorithm is "BCABAAAA".

2. MTF.

Initially, each possible byte value is written to a list (alphabet), in a cell with a number equal to the byte value, i.e. (0, 1, 2, 3,..., 255). This list changes as the data is processed. As the next character arrives, the number of the element containing its value has sent to the output. After that, this symbol moves to the beginning of the list, shifting the remaining elements to the right.

Modern algorithms (for example, bzip2) use the BWT algorithm before the MTF algorithm, so as an example, consider the string S = "BCABAAAA" obtained from the string "ABACABAA" as a result of the Burroughs-Wheeler transformation (more on it later). The first character of the string S = "B" is the second element of the alphabet "ABC", so the output is one. After moving 'B' to the beginning of the alphabet, it takes the form "BAC". Further work of the algorithm:

Table 1 — MTF conversion algorithm:

Symbol	List	Output
B	ABC	1
C	BAC	2
A	CBA	2
B	ACB	2
A	BAC	1
A	ABC	0
A	ABC	0
A	ABC	0

Thus, the result of the MTF(S) algorithm is “12221000".

3. ARI.

Trying on arithmetic coding we get:

ARI(S) =101110100111101001001000

Thus, if we are dealing with eight-bit characters, then the input is 8*8=64 bits, and the output is 24, that is, the compression ratio 62.5%.

Consider the same example, but with the addition JBE - BWT+MTF+JBE+ARI:

Points 1 and 2 are the same.

4. JBE.

Table 2-Algorithm for applying JBE encoding

Original		Data 1		Temporary byte data		Data 2	
1	00000001	1	00000001	1	00000001	248	11111000
2	00000010	2	00000010	1	00000001	-	-
2	00000010	2	00000010	1	00000001	-	-
2	00000001	2	00000010	1	00000001	-	-
1	00000000	1	00000001	1	00000001	-	-
0	00000000	-	-	0	00000000	-	-
0	00000000	-	-	0	00000000	-	-
0	00000000	-	-	0	00000000	-	-

At the output, we have a record of the original input length + Data record I + Data record II =24812221.

5. ARI.

Trying on arithmetic coding we get:

ARI(S) =10111011011001100110110

Thus, if we are dealing with eight-bit characters, then the input is 8*8=64 bits, and the output is 24, that is, the compression ratio 62.5%.

Result

Figure 4 shows that 8-bit bitmap images have compressed with good compression ratio by algorithms that combined with J-bit encoding.

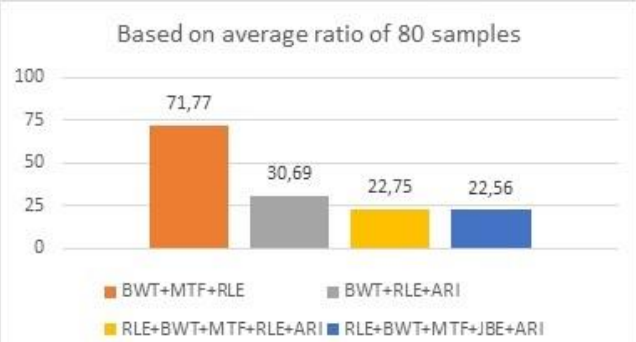

Figure 4. Diagram of the ratio of compression of 8-bit bitmaps between different algorithms

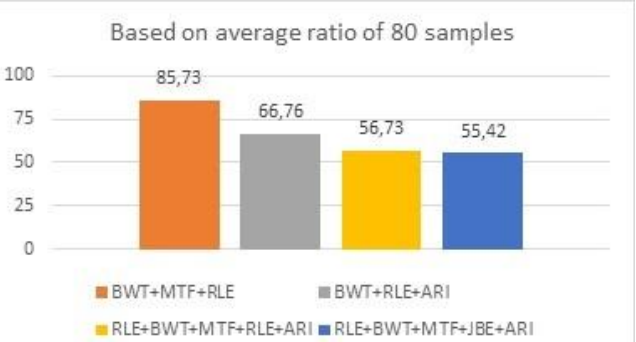

Figure 5. Diagram of the ratio of compression of 24-bit bitmaps between different algorithms

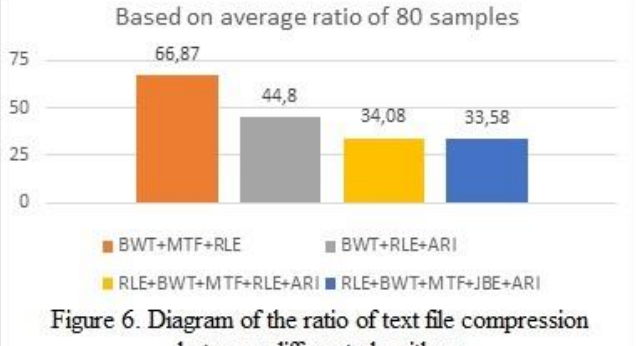

Figure 6. Diagram of the ratio of text file compression between different algorithms

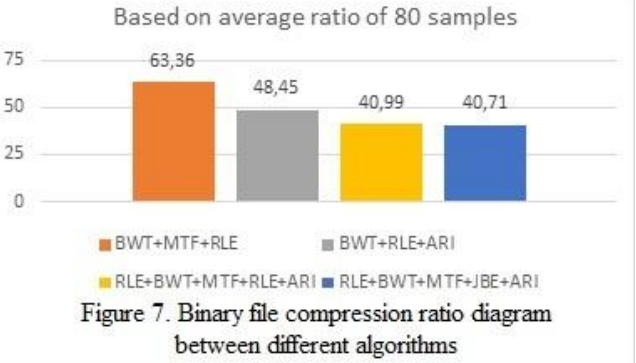

Figure 7. Binary file compression ratio diagram between different algorithms

Figure 5 shows that 24-bit bitmap images has compressed with better compression ratio by algorithms that combined with J-bit encoding. A 24-bit bitmap image has more complex data than 8 bits since it is storing more color. Loss compression for image would be more appropriate for 24-bit

bitmap image to achieve best compression ratio, even thought that will decrease quality of the original image.

Figure 6 shows: text files have compressed with better compression ratio by algorithms that combined with J-bit encoding.

Figure 7 show: binary files have compressed with better compression ratio by algorithms that combined with J-bit encoding.

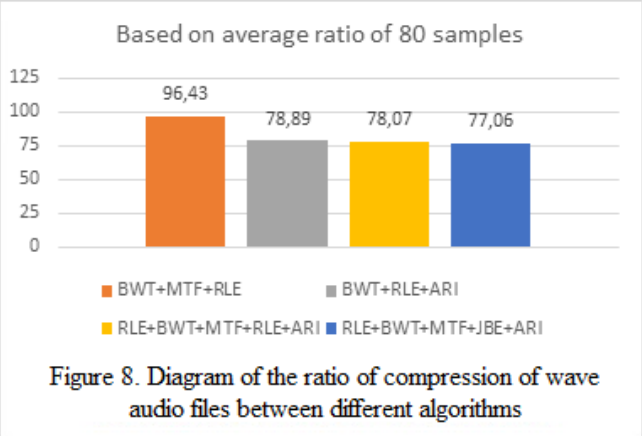

Figure 8. Diagram of the ratio of compression of wave audio files between different algorithms

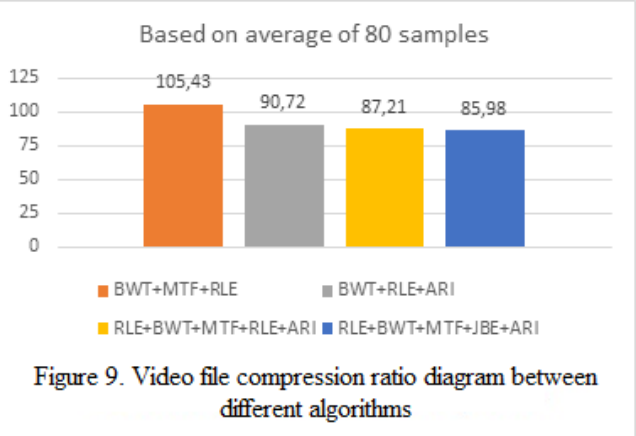

Figure 9. Video file compression ratio diagram between different algorithms

Figure 8 shows: wave audio files have compressed by better compression ratio by algorithms that combined with J-bit encoding.

Figure 9 shows: video files have compressed by the best compression ratio with the combined J-bit encoding algorithms.

CONCLUSION

The modified data compression algorithm was proposed. The experiment has conducted. It has used six file types with 80 different sizes for each file type. As a result, 4 combinational algorithms have tested and compared. The proposed algorithm shows a better compression ratio after to insert between" forward motion transformation" (MTF) and arithmetic encoding (ARI). The study provides both the theoretical part and practical examples. The considered algorithm has the prospect of introducing other data compression algorithms into the structure.

REFERENCES

1. Capo-chichi, E. P., Guyennet, H. and Friedt, J. K-RLE a New Data Compression Algorithm for Wireless Sensor Network. In Proceedings of the 2009 Third International Conference on Sensor Technologies and Applications.
2. Salomon, D. 2004. Data Compression the Complete References Third Edition. Springer-Verlag New York, Inc.
3. Nelson, M. 1996. Data compression with Burrows-Wheeler Transform. Dr. Dobb's Journal.
4. Campos, A. S. E. Run Length Encoding. Available: http://www.arturocampos.com/ac_rle.html (last accessed July 2012).
5. Campos, A. S. E. Move to Front. Available: http://www.arturocampos.com/ac_mtf.html (last accessed July 2012).
6. Campos, A. S. E. Basic arithmetic coding. Available: http://www.arturocampos.com/ac_arithmetic.html (last accessed July 2012).
7. Springer, Handbook of Data Compression Fifth Edition.
8. Agus Dwi Suarjaya. Information Technology Department Udayana University. Bali, Indonesia. A New Algorithm for Data Compression Optimization.