An Efficient Text Compression for Massive Volume of Data

M.Baritha Begum
Assistant professor
Saranathan College of Engineering
Trichy, Tamilnadu

Dr.Y.Venkataramani
Principal
Saranathan College of Engineering
Trichy, Tamilnadu

ABSTRACT
To propose a new text compression technique for ASCII texts for the purpose of obtaining good performance on various document sizes. This algorithm is composed of two stages. In the first stage, the input strings are converted into the dictionary based compression. In the second stage, the redundancy of the dictionary based compression is reduced by Burrows wheeler transforms and Run length coding. The algorithm has good compression ratio and reduces bit rate to execute the text with increase in the speed.

General Terms
Lossless text compression, Dictionary making algorithm

Keywords
Dictionary Based Encoding (DBE), Burrows-Wheeler Transform (BWT), Run Length Encoding (RLE).

1. INTRODUCTION
In the field of computer science and theory, Data compression is essential to evolve a method to compress a lengthy data or to reduce the Bit-rate .This method involves the process of encoding information using fewer bits than the original representation .This process of compression reduces the consumption of resources of data storage or transmission bandwidth. However, any compressed data is required to be decompressed to be used which may cause detrimental effect in application. The design of data compression schemes involves trade-offs among various factors, including the degree of compression, the amount of distortion introduced and the computational resources are required to compress and uncompress the data.

Lossless data compression is a class of data compression algorithms. It enables the process of reconstructing exact original data from the compressed data. But in lossy data compression, the reconstruction of approximate original data leads to a better compression ratio. Original and decompressed data being identical is important to use lossless data compression.

Nowadays the need for the most efficient data compression algorithms has been increased in recent years. The implementation complexity, the large execution time and the large memory size are needed in the compression and decompression algorithms because of the large number of symbols in the alphabet of the original information sources. [1] Concerning compression ratio, one of the best compression algorithms in practice is dictionary based text compression.

The rest of paper is organized as follows. Section 2 gives background on text transformation algorithms. Section 3 presents dictionary making algorithm. Section 3.1 presents Burrows wheeler transform. Section 3.2 presents Run length coding. Section 3.3 presents decoding. Section 4 provides performance analysis. Section 5 concludes the paper.

2. BACKGROUND ON TEXT COMPRESSION ALGORITHMS
Any real number between one and zero is encoded as a message in the arithmetic coding. Arithmetic coding typically has a better compression ratio than Huffman coding, as it produces a single symbol rather than several separate code words. Arithmetic coding is a lossless coding technique. There are a few disadvantages of arithmetic coding. Is One of the disadvantage in this coding is that symbol can be decoded after receiving the whole code word. But if a bit from the codeword is corrupted, the entire message will be corrupted. Another is that there is a limit to the precision of the number which can be encoded, thus limiting the number of symbols to encode within a codeword. There also exist many patents upon arithmetic coding, so the use of some of the algorithms also call upon royalty fees. [4, 5]

In Huffman coding encode symbols with higher probability will appear only minimum output namely a fewer bits. Each symbol is represented as different amount of bits. No symbol code will act as prefix for another symbol code. So the code which has this property is called Prefix code. The drawback of Huffman code is that maintaining structures with full Huffman Codes requires limited code length to keep a code as an integer with limited value. This can be achieved with some loss in compression ratio. But with possible simplicity of structures, speed of calculations can be a good reason to choose this coding. [10, 11]

LZ77 uses the built in implicit assumption that patterns in the input data occurs close together. Data streams that don't satisfy this assumption compress poorly. Another disadvantage of LZ77 is the limited size L of the Look ahead buffer. The size of matched strings is limited to L-1,but L must be kept small, since the process of matching strings involves comparing individual symbols .If L were doubled in size, compression would improve, since longer matches would be possible, but the encoding would be much slower while searching for long matches. The size S of the search buffer is also limited. [5]

Figure.1 illustrates the dictionary based text compression algorithm. The original text file is provided as input to the transformation by dictionary based compression, which outputs the transformed text. This output is provided to an existing, data compression algorithm (such as BWT with RLE), which compresses the transformed text. To decompress, one merely reverses this process, by first invoking the appropriate decompression algorithm, and then providing the resulting text to the inverse transform.

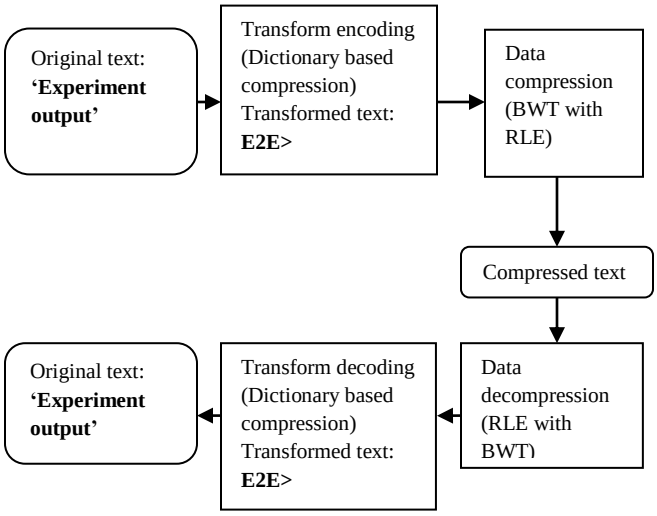

Figure 1: dictionary based text compression incorporating a lossless, reversible transformation

One well-known example of the text compression is the Burrows-Wheeler Transform (BWT) [7] outlined in figure 1. BWT combines with run length encoding [8, 9, 10, and 11] to provide one of the best compression ratios available on a wide range of data. However, none of these methods has been able to reach the theoretical best-case compression ratio consistently. Dictionary Based text compression approach is a new compression algorithm developed while trying to attain better compression ratio.

3 .DICTIONARY ALGORITHM

We extract the words mainly from Calgary Corpus files [12] with 6, 18,108 numbers of words. The words having Upper case letters are converted into words with lower case letters.

During the formation of table with words, occurrences of words are checked, words occurring frequently are identified and they are set in descending order according to their frequency. In this method, it would generate 8900 words. In the first assignment, ASCII characters are assigned for each word as a code.

!@#$%^&*()_+..... . Upto ASCII character of 255

For the remaining words from table, all ASCII characters are combined with capital letter "A"-"Z" and assigned as follows to get permutation 1.

A! A@ A# A$ A% A^ A& A* A (A) A_ A+……upto ASCII character of 255.

B! B@ B# B$ B% B^ B& B* B (B) B_ B+……. upto ASCII character of 255.

Z! Z@ Z# Z$ Z% Z^ Z& Z* Z (Z) Z_ Z+ …….. ..Upto ASCII character of 255.

For the remaining words from the table with permutation 1, all ASCII characters are combined with capital letter "A"-"Z" and assigned as follows to get permutation 2

AA! AA@ AA# AA% AA^ AA& AA* AA (AA) AA_ AA+ …. Upto ASCII character of 255

BB! BB@ BB# BB$ BB% BB^ BB& BB* BB (BB) BB_ BB+…. upto ASCII character of 255 ……

ZZ! ZZ@ ZZ# ZZ$ ZZ% ZZ^ ZZ& ZZ* ZZ (ZZ) ZZ_ ZZ+ ……. upto ASCII character of 255.

The shortest code is assigned to most frequently used words. The longest code is assigned to less frequently used words.

3.1 BWT Transform

The BWT is an algorithm that takes a block of data and rearranges it using a sorting algorithm. The resulting output block contains exactly the same data elements that it starts with, differing only in their ordering. The transformation is reversible; meaning the original ordering of the data elements can be restored with no loss of fidelity.

The BWT is performed on an entire block of data at once. Most of today's familiar lossless compression algorithms operate in streaming mode, reading a single byte or a few bytes at a time. But with this new transform, we want to operate on the largest chunks of data possible. Since the BWT operates on data in memory, it encounters files too big to process in one full swoop. In these cases, the file must be split up and processed a block at a time. The output result from this proposal of the dictionary based compression transform is given to the BWT.

3.2 Run Length Coding

Run Length Encoding (RLE) is a simple and popular data compression algorithm. It is based on the idea to replace a long sequence of the same symbol by a shorter sequence. It includes some interesting aspects for the RLE field too. An important property of the output of the BWT is the presents of many runs, which results in overestimating the probability of symbols outside the run. This problem is called "Pressure of Runs" and can be decreased by RLE. Besides that, a modification of the MTF algorithm is described which moves the next symbol to the second place in the list instead of the first place, except the old position of the new symbol was 0 or 1. If the old position of the new symbol is 1, it is moved to the first place only if the last output number was different from 0.

The output of the BWT is applied to the RLE. The implementation is simple. If successive bytes are equal, note the frequency of its occurrence, then output the count value, and continue encoding, of course to discard the repeated bytes. If the bytes are not equal, then output the first, make the second the first, and get the next byte as a second, and start again.

3.3 Encoding Algorithm

Extract words from input document are compared with dictionary table words. If there is a match, it selects the corresponding code. In this way the whole document is converted into new coding format.

Example, a section of text from Calgary corpus paper looks like this in the original text:

"Speech is our everyday, informal, communication medium. But although we use it a lot, we probably don't assimilate as much information through our ears as we do through our eyes, by reading or looking at pictures and diagrams. You go to a technical lecture to get the feel of a subject\ (em the overall arrangement of ideas and the motivation behind them\ (em and fill in the details, if you still want to know

them, from a book. You probably find out more about the news from ten minutes with a newspaper than from a ten-minute news broadcast"

Number of characters required=545

Running this text through the dictionary based encoder yields the following text:

N6~CCmv2!mF<AAA~C<mlA~E1C0F95!CV5A,D7!D,LS7I 9,PEË9N*,|qBB5Ý¡!qFAÁN¡µEu!Ò¡!NMÕ,úÇ:)¨□¨.)þÿ¸#öv; kø+"Ó·ÇÙ□Á□Aê

Number of characters required=89

3.4 Decoding Algorithm

The decoding is easier than the encoding. The final encoding output is taken and fed to run length decoder. Run length decoder output is applied to the BWT decoder. In BWT decoder output, upper case letters followed by single ASCII character is identified as a code. If upper case letters are followed by two ASCII characters, the second ASII character is identified as separate code. Extracted code is compared with dictionary table and corresponding words are collected in the output file. This output file after processing looks the same as the initial document since the compression and decompression is lossless.

4. PERFORMANCE ANALYSIS

From these experiments on the transformation algorithms mentioned in section 3 using standard Calgary Corpus text file collections. [12]

The performance issue such as compression ratio and Bits Per Character (BPC) are compared for the seven cases i.e., simple Arithmetic coding, Huffman with BWT, LZSS with BWT, Dictionary based Encoding (DBE), Dictionary based Encoding with BWT, Dictionary based Encoding with RLE and Dictionary based Encoding with BWT and RLE. The results are shown graphically and it is proved that DBE with BWT and RLE performs better than all other techniques in compression ratio, Bits per Character (BPC).

$$\text{Compression ratio} = \frac{\text{Output file size}}{\text{Input file Size}}$$

$$\text{Bits per character (BPC)} = \frac{\text{Output file size}}{\text{Input file size}} * 8$$

Table 1: List of files used in experiments

File name	Size(byte)	Description
Bib	111,261	Bibliography
Geo	102,400	Geological seismic data
Obj1	21,504	VAX object program
paper1	53,161	Technical Paper
Paper2	82,199	Technical Paper
Paper3	46,526	Technical Paper
Paper4	13,286	Technical Paper
Paper5	11,954	Technical Paper
Paper6	38105	Technical Paper
Progc	39,611	Source Code in "C"
Progl	71,646	Source Code in "Pascal"
Progp	49,379	Text: English Text

Fig.1: BPC comparison for Calgary corpus files

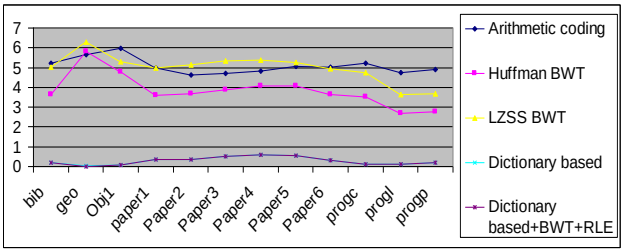

Fig.2: Compression Ratio Comparison for Calgary Corpus Files

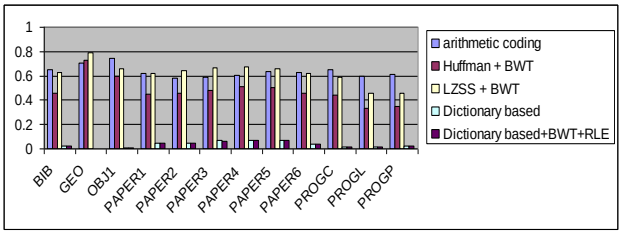

Table.2: BPC comparison for Calgary corpus files.

File name	Arithmetic coding	Huffman+BWT	LZSS+BWT	DBE	DBE+BWT	DBE+RLE	Dictionary based+BWT+RLE
Bib	5.232	3.656	5.016	0.2	0.2	0.2	0.192
Geo	5.656	5.8	6.304	0.024	0.024	0.024	0.016
Obj1	5.968	4.768	5.288	0.08	0.08	0.08	0.08
paper1	4.984	3.616	4.976	0.36	0.36	0.352	0.352
Paper2	4.624	3.68	5.136	0.352	0.352	0.352	0.344
Paper3	4.712	3.856	5.336	0.528	0.528	0.528	0.512
Paper4	4.824	4.064	5.376	0.584	0.592	0.584	0.584
Paper5	5.064	4.056	5.256	0.552	0.552	0.552	0.552
Paper6	5.008	3.632	4.952	0.32	0.32	0.32	0.312
Progc	5.24	3.504	4.728	0.12	0.12	0.12	0.112
Progl	4.76	2.68	3.648	0.136	0.136	0.136	0.128
Progp	4.896	2.76	3.688	0.2	0.2	0.2	0.2

Table 3 Comparison of Compression Ratio

File name	Arithmetic coding	Huffman+BWT	LZSS+BWT	DBE	DBE+BWT	DBE+RLE	DBE+BWT+RLE
Bib	0.654	0.457	0.627	0.025	0.025	0.025	0.024
Geo	0.707	0.725	0.788	0.003	0.003	0.003	0.002
Obj1	0.746	0.596	0.661	0.010	0.010	0.010	0.010
paper1	0.623	0.452	0.622	0.045	0.045	0.044	0.044
Paper2	0.578	0.460	0.642	0.044	0.044	0.044	0.043
Paper3	0.589	0.482	0.667	0.066	0.066	0.066	0.064
Paper4	0.603	0.508	0.672	0.073	0.074	0.073	0.073
Paper5	0.633	0.507	0.657	0.069	0.069	0.069	0.069
Paper6	0.626	0.454	0.619	0.040	0.040	0.040	0.039
Progc	0.655	0.438	0.591	0.015	0.015	0.015	0.014
Progl	0.595	0.335	0.456	0.017	0.017	0.017	0.016
Progp	0.612	0.345	0.461	0.025	0.025	0.025	0.025

5. CONCLUSION

This paper proposes a method of text transformation using Dictionary based encoding. In a channel, the reduction of transmission time is directly proportional to the amount of compression. If the input text is replaced by variable length codes with its length less than its average size, the size of input text can be reduced by using dictionary based compression, BWT and RLE. And if the coding scheme used is a highly redundant coding scheme, this would increase the redundancy in the input text. In this method, an experiment of transforming the standard Calgary corpus data files by using various algorithms was conducted. This method achieves good compression ratio and reduces bits per character.

6. REFERENCE

[1] G.Hold and T.R Marshall, Data compression, John Wiley, New York 1991.

[2] Jirapond Tadrat and Veera Boonjing, 2008"An Experiment study on Transformation for Compression using stop lists and Frequent words" IEEE Transactions on information technology.

[3] Data compression: the complete reference By David Salomon

[4] A.carus, A.Mesut, 2010,"Fast text compression using Multiplies dictionaries", Information technology journal 9(5) 1013-1021.

[5] M. Burrows and D. J. Wheeler. "A Block-sorting Lossless Data Compression Algorithm", SRC Research Report 124, Digital Systems Research Center.

[6] J.L. Bentley, D.D. Sleator, R.E. Tarjan, and V.K. Wei, " A Locally Adaptive Data Compression Scheme", Proc. 22nd Allerton Conf. On Communication, Control, and Computing, pp. 233-242, Monticello, IL, October 1984, University of Illinois

[7] J.L. Bentley, D.D. Sleator, R.E. Tarjan, and V.K. Wei, "A Locally Adaptive Data Compression Scheme", Commun. Ass. Comp. Mach., 29:pp. 233-242, April 1986.

[8] R.G. Gallager. "Variations on a theme by Huffman", IEEE Trans. Information Theory, IT-24(6), pp.668-674, Nov, 1978

[9] D.A.Huffman. "A Method for the Construction of Minimum Redundancy Codes", Proc. IRE, 40(9), pp.1098-1101, 1952

[10] Nelson C. Francisco, Nuno M. M. Rodrigues, Eduardo A. B. da Silva, Murilo Bresciani de Carvalho, Sergio M. M. de Faria, , October 2010 "Scanned Compound Document Encoding Using Multiscale Recurrent Patterns" IEEE transactions on image processing, vol. 19, no. 10.

[11] Umesh S. Bhadade Prof. A.I. Trivedi, January 2011 "Lossless Text Compression using Dictionaries", International Journal of Computer Applications (0975 – 8887) Volume 13– No.8.

[12] Compression test results, corpus.canterbury.ac.nz/