BURROWS WHEELER BASED DATA COMPRESSION AND SECURE TRANSMISSION

M.P.Bhuyan[1], V.Deka[2], S.Bordoloi[3]

[1]*Department of Information Technology, Gauhati University*
[2]*Department of Information Technology, Gauhati University*
[3]*Department of Computer Applications, Assam Engineering College*

Abstract

Now days, computer technology mostly focusing on storage space and speed With the rapid growing of important data and increased number of applications, devising new approach for efficient compression and encryption methods are playing a vital role in performance. In this work, burrows wheeler transformation is introduced for pre processing of the input data and made several performance analysis experiments over different compression techniques for various types of text files and improved compression ratio has been found by applying burrows wheeler transform as pre-processing step.

Keywords: *RLE, arithmetic, Huffman, LZW, lossless data compression.*

---***---

1. INTRODUCTION

It is seen that day by day amount data increases, so it becomes necessary to optimize the storage space for efficient utilization. In lossy compression methods, during the decompression process it is not possible to recover the original file. To overcome this difficulty we need lossless data compression technique. Use of lossless data compression technique reconstructs the original file as it was before the compression. Lossless data compression is an important compression technique to compress text files, because removal of much redundancy in text files causes the change in meaning of the original data or text. Burrows Wheeler Transform is widely used in all over the world for lossless data compression [1]. People have devoted lots of time in innovating new techniques for the enhancement of lossless data compression algorithm. We will try to use Burrows Wheeler transform in lossless image compression and if possible special care can be taken. Further if possible, we will try to use the technique in the compression of other kind of files like audio or video files. The rest of the paper is organized as follows: Section 2 presents a brief explanation about Burrows Wheeler Transformation; Section 3 discusses about some existing lossless data compression algorithms, Section 4 reflects the challenges of combining cryptography and compression and the cryptographic algorithm, Section 5 has its focus on comparing the performance of different lossless data compression algorithms, finally, section 6 concludes the work and section 7 proposes the future work.

2. BURROWS WHEELER TRANSFORM

Burrows Wheeler Transform is a transformation technique first introduced in 1994[6], which is the unpublished work by Wheeler in 1983. The fundamental concept behind this technique is that when a text file or a character string is transformed the size of the string does not change. The transformation only permutes the string into n permutations, where n is the total number of characters in the string. After performing Burrows Wheeler Transform new transformed string can be compressed easily with compression method like run length encoding. In addition, if move to front encoding is applied to the transformed data then it can be compressed quite efficiently.

2.1 The Forward Transform

Consider a string p= dckdacm.
Step1: The original sequence p is copied to the first row, also referred to as index 0. The sequence is then sorted with all left cyclic permutations into each next index row. The step 1 of the BWT is presented in Table 2.1.1

Step2: The rows are sorted lexicographically then from this output sequences. The step 2 of the BWT is shown in Table 2.1.2. Step 3 is the final step of the BWT process consisting of output of the BWT and the final index.

Step3: The original sequence p= dckdacm appears in the fifth row of Table 2.1.2, and the output of the BWT transform is the last column, indicated by L = ddakmcc which is shown in table2.1.3.

With the index = 4, the result can be written as BWT = [index, L], where L is the output of the Burrows-Wheeler transform and index indicates the location of the original sequence in the lexicographically ordered sequence. We also determine the first column F = accddkm, which can be obtained from L by sorting which is required reverse transform of the BWT.

Table 2.1.1 step1 of BWT with all cyclic permutations

Index	Step1 Output
0	dckdacm
1	ckdacmd
2	kdacmdc
3	dacmdck
4	acmdckd
5	cmdckda
6	mdckdac

Table 2.1.2 step2 of BWT rows of step1 are lexicographically sorted

Index	Step3 Output
0	d
1	d
2	a
3	k
4	m
5	c
6	c

Table 2.1.3 step3 of BWT contains last character of each row of step1

Index	Step2 Output
0	acmdckd
1	ckdacmd
2	cmdckda
3	dacmdck
4	dckdacm
5	kdacmdc
6	mdckdac

2.2 The Reverse Burrows-Wheeler Transform:

The BWT is a reversible transformation which can recover the original sequence from the BWT output sequence. In reverse transform only the BWT output sequence L and index are needed for reconstructing the original sequence. To solve the reverse BWT using output of the BWT, L and index, the reverse BWT is presented in Table 2.2.1.

Input (ddakmcc, index=4).
Output=dckdacm

Table 2.2.1 Step1 of reverse BWT

Index	Reverse BWT input	Previous combine	Sort	New combine (BWT i/p+ Sort)
0	d	Φ	a	da
1	d	Φ	c	dc
2	a	Φ	c	ac
3	k	Φ	d	kd
4	m	Φ	d	md
5	c	Φ	k	ck
6	c	Φ	m	cm

Table 2.2.2 Step2 of reverse BWT

Index	Reverse BWT input	Previous combine	Sort	New combine (BWT i/p+ Sort)
0	d	da	ac	dac
1	d	dc	ck	dck
2	a	ac	cm	acm
3	k	kd	da	kda
4	m	md	dc	mdc
5	c	ck	kd	ckd
6	c	cm	md	cmd

Table 2.2.3 Step3 of reverse BWT

Index	Reverse BWT input	Previous combine	Sort	New combine (BWT i/p+ Sort)
0	d	dac	acm	dacm
1	d	dck	ckd	dckd
2	a	acm	cmd	acmd
3	k	kda	dac	kdac
4	m	mdc	dck	mdck
5	c	ckd	kda	ckda
6	c	cmd	mdc	cmdc

Table 2.2.4 Step4 of reverse BWT

Index	Reverse BWT input	Previous combine	Sort	New combine (BWT i/p+ Sort)
0	d	dac	acm	dacm
1	d	dck	ckd	dckd
2	a	acm	cmd	acmd
3	k	kda	dac	kdac
4	m	mdc	dck	mdck
5	c	ckd	kda	ckda
6	c	cmd	mdc	cmdc

Table 2.2.5 Step5 of reverse BWT

Index	Reverse BWT input	Previous combine	Sort	New combine (BWT i/p+ Sort)
0	d	dacmd	acmdc	dacmdc
1	d	dckda	ckdac	dckdac
2	a	acmdc	cmdck	acmdck
3	k	kdacm	dacmd	kdacmd
4	m	mdckd	dckda	mdckda
5	c	ckdac	kdacm	ckdacm
6	c	cmdck	mdckd	cmdckd

Table 2.2.6 Step6 of reverse BWT

Index	Reverse BWT input	Previous combine	Sort	New combine (BWT i/p+ Sort)
0	d	dacmdc	acmdck	acmdckd
1	d	dckdac	ckdacm	ckdacmd
2	a	acmdck	cmdckd	cmdckda
3	k	kdacmd	dacmdc	dacmdck
4	m	mdckda	dckdac	dckdacm
5	c	ckdacm	kdacmd	kdacmdc
6	c	cmdckd	mdckda	mdckdac

Burrows Wheeler transformation is used in various field of research from the beginning of its invention, it is seen that the importance of Burrows Wheeler Transform is increased day by day. It is widely used as a pre processing stage in lossless data compression. BWT is used to compress DNA sequencing in the field of bioinformatics. Research is going on to compress the medical information or some astronomical images. Due to the efficiency of BWT, researchers from various backgrounds are attracted to this technique and use this technique to optimize their resources. So many people try to improve the BWT technique for better processing in their own fields. In the coming days, this technique may overcome the deficiency of storage space.

3. SOME LOSSLESS DATA COMPRESSION ALGORITHMS

3.1 Run Length Encoding

Run Length Encoding (RLE) compression technique is used when a given file contains too many redundant data or long run of similar characters. The repeated string or characters present in the input file or message is called a run which is encoded into two bytes. The first byte represents the value of the character in the run and the second byte contains the number of times given character appears in the run. For example the following string can be represented in RLE as

ZZZZZZkkkHHHHHttt
Z6k3H5t3

3.2 Huffman Coding

Huffman coding is a data compression technique in which each input character is replaced with variable length binary digits which are called codeword and the codeword has been derived in a particular way based on the probability of occurrence of each symbol or character. The most frequent symbols in the source have the shortest length code and the least frequent symbol has the longest code. This technique is implemented by creating a binary tree of nodes. These can be stored in data structures like array or link list, the size of which depends on the number of symbols, n.

3.2 Arithmetic Coding

The arithmetic coding concept is to have a probability value 0 to 1, and assign to every symbol a range in between 0 and 1 based on its probability, higher the probability, higher is the range. Once we have defined the ranges and the probability, encoding of symbols can be started, every symbol in encoding process gives us a new floating point range to encode the next symbol.

3.4 LZW Coding

This is a dictionary based compression algorithm which is implemented by depending on a dictionary. A dictionary is a collection of some possible words of a particular language and is stored in tabular fashion, some indexes are used to represent repeating symbols. In Lempel-Ziv Welch algorithm or LZW, one kind dictionary is used to store the symbols. In the compression process, the index values are used in place of the similar repeated strings or symbols. Creation of dictionary is a dynamic process, so it is not transferred with the encoded data, during decompression the dictionary is created automatically.

4. CHALLENGES OF COMBINING COMPRE-SSION AND CRYPTOGRAPHY AND THE CRYPTOGRAPHIC ALOGORITHM USED IN THIS ANALYSIS

Most of the powerful cryptographic algorithms increase size after encryption and if we modify the algorithm then strength of the algorithm decreases. So we need algorithms of the type stream cipher, substitution cipher etc. One substitution type algorithm is given below.

Encryption Algorithm:

Step 1: Input Text (T)
Step 2: Check ASCII value of each character
Step 3: If (Character (ASCII value)>127)

```
        ASCII value=255-ASCII value of the character
    Else
        ASCII value=127+ASCII value of the character
Step 4: Now print the corresponding character of the ASCII value in ENCRYPT.txt file.
```

Decryption Algorithm:

```
Step1: Check each character of the file ENCRYPT.txt
Step 2: If (Character (ASCII value)>127)
        ASCII value= ASCII value-127
    Else
        ASCII value=255-ASCII value of the character
Step 3: Now print the corresponding character of the ASCII value in DECRYPT.txt.
```

5. PERFORMANCE ANALYSIS

The performance analysis of the compression algorithms are done for different text files of different size. Comparison is done in terms of compression ratio which is the ratio of the size of file after compression to the size of file before compression. The comparison is also shown graphically. Text files of different size with different characters have been taken and then compression is done in MATLAB for the different compression algorithms. The results are shown in the table 5.1 and 5.2. The following charts showing the comparison graphically for different algorithms with respect to different text files.

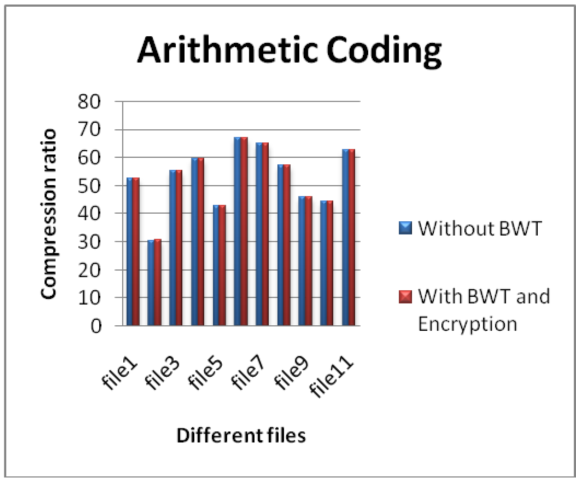

Fig 5.1 Comparison of compressed files in Arithmetic Coding

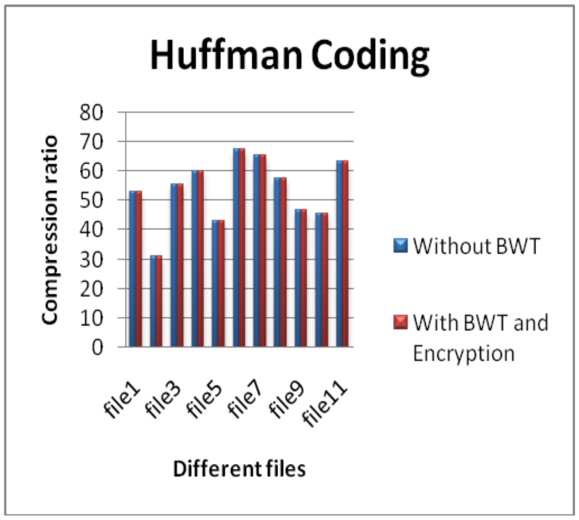

Fig 5.2 Comparison of compressed files in Huffman Coding

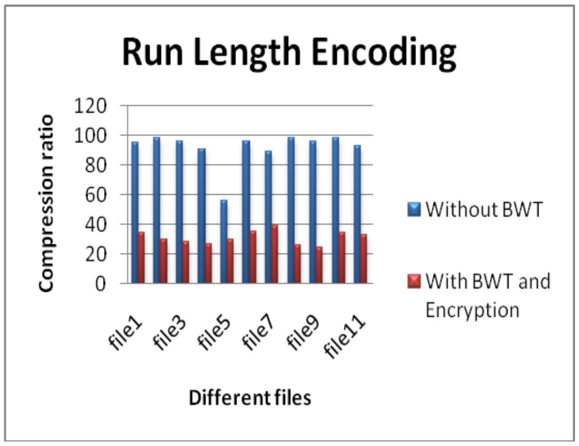

Fig 5.3 Comparison of compressed files in Run Length Encoding

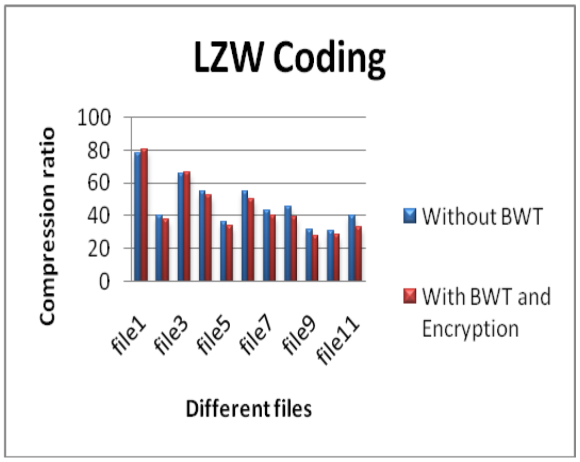

Fig 5.4 Comparison of compressed files in LZW Coding

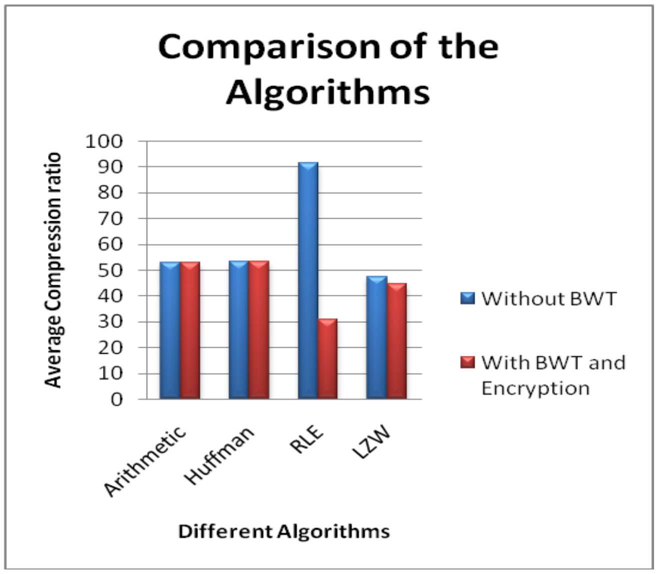

Fig 5.5 Comparison of Average Compression ratio of different algorithms

From the analysis it is seen that RLE compression has shown drastic improvement and LZW has shown a little improvement in the compression ratio after preprocessing with Burrows Wheeler Transform and encryption. Though the other compression algorithms have not shown much improvement in compression ratio after preprocessing with Burrows Wheeler Transform For the time being RLE has performed much better with pre-processing with Burrows Wheeler Transform and encryption.

6. CONCLUSIONS

After doing a detail study on different compression algorithms it is seen that RLE shows a great improvement on compression ratio after preprocessing with Burrows Wheeler Transformation. With these results we can conclude that the combination of Burrows Wheeler Transform and RLE gives us a best compression method for lossless data compression.

FUTURE WORK

In future it is possible to apply the same technique to compress image, audio and video files etc. It is possible to make the encryption process powerful by using some powerful algorithms like Blowfish, RC5 etc.

REFERENCES

[1]. B. Balkenhol and S Kurtz. Universal Data Compression Based on the Burrows-Wheeler Transformation: Theory and Practice. IEEE transactions on Computers,49(10):1043-1053, October 2000.

[2]. J. Abel. Improvements to the Burrows-Wheeler Compression Algorithm: After Bwt Stages. In ACM Trans. Computer Systems, May 2003

[3]. M. Burrows and D. J. Wheeler. A Block-Sorting Lossless Data Compression Algorithm Tech. Rep. 124, Digital Systems Re-search Centre, 1994.

[4]. M. Nelson. Data Compression With the Burrows-Wheeler Transform. In Dr. Dobbs Journal,, pages 746-751, September 1996.

[5]. Bitla Srinivas and V K Govindan. A Modified Approach to the New Lossless Data Compression Method International Journal of Computer Communication Technology (IJCCT), 2(7):44-48, 2011.

[6]. C. Sidney Bums Haitao Guo. Waveform and Image Compression Using the Burrows Wheeler Transform and the Wavelet Transform Proceedings, International Conference on Image Processing, 1:65-68, 1997.

[7]. J. Ziv and A. Lempel. A Universal Algorithm for Sequential Data Compression IEEE Transactions on Information Theory IT-23(3):337-343, May 1977.

[8]. K. Holtz and E. Holtz. Lossless Data Compression Techniques WESCON/94 Conference, Session W23, Advanced Information Management, Anaheim Convention Centre, Anaheim, California,, pages 1043-1053, September 1994.

[9]. J. Gilchrist and A. Cuhadar. Parallel Lossless Data Compression Based on Burrows-Wheeler Transform. 1st International Conference on Advanced Networking and Applications (AINA'07), pages 1043-1053, May 2007

[10]. J. Ziv and A. Lempel. Compression of Individual Sequences Via Variable Rate Coding. IEEE Transactions on Information Theory, IT- 24(5):530-535, September 1978.

[11]. R.E. Tarjan J.L. Bentley, D.D. Sleator and V.K. Wei. A Locally Adaptive Data Compression Algorithm Communications of the ACM, 29(4):320-330, April 1986.

[12]. DI Michael Schindler. A Fast BlockSorting Algorithm for Lossless Data Compression

[13]. Jer Min Jou and Pei-Yin Chen. A Fast and Efficient Lossless Data Compression Method IEEE TRANSACTIONS ON COMMUNICATIONS, 47(9), September 1999.

[14]. M. Effros. Universal Lossless Source Coding With the Burrows Wheeler Transform. Data Compression Conf., Snowbird, UT, pages 178-187, March 1999

[15]. Guy E. Blelloch. Introduction to Data Compression. Computer Science Department, Carnegie Mellon University, September 2010.

Table 5.1 Size of different files before and after compression and their compression ratio

File	Original Size(bytes)	Compressed Size(bytes)				Compression ratio			
		Arithmetic	Huffman	RLE	LZW	Arithmetic	Huffman	RLE	LZW
File1	134	70	71	127	105	52.52	53.00	94.78	78.36
File2	298	90	92	293	120	30.45	30.87	98.32	40.27
File3	452	251	250	435	296	55.37	55.30	96.24	65.48
File4	892	530	534	810	489	59.47	59.86	90.81	54.82
File5	1092	468	469	611	396	42.86	42.95	55.95	36.26
File6	1937	1300	1308	1856	1059	67.12	67.53	95.82	54.67
File7	2861	1859	1868	2539	1242	64.99	65.29	88.75	43.41
File8	25210	14419	14496	24752	11518	57.40	57.71	98.53	45.69
File9	42080	19360	19688	40232	13099	46.00	46.79	95.61	31.13
File10	50464	22439	22864	49544	15321	44.47	45.31	98.18	30.36
File11	60456	38006	38272	55912	24031	62.87	63.31	92.48	39.75
Avg.	**16897.82**	**8981.09**	**9082.91**	**16101.00**	**6152.36**	**53.05**	**53.45**	**91.41**	**47.29**

Table 5.2 Size of different files before and after compression and their compression ratio after preprocessing with BWT and encryption

File	Original Size(bytes)	Compressed Size(bytes)				Compression ratio			
		Arithmetic	Huffman	RLE	LZW	Arithmetic	Huffman	RLE	LZW
File1	134	70	71	46	108	52.61	52.98	34.32	80.60
File2	298	91	92	88	112	30.49	30.87	29.53	37.58
File3	452	251	250	128	299	55.34	55.30	28.31	66.15
File4	892	530	534	240	468	59.49	59.86	26.90	52.47
File5	1092	469	471	324	369	42.88	43.05	29.67	33.79
File6	1937	1301	1309	689	970	67.13	67.51	35.57	50.08
File7	2861	1859	1868	1120	1145	64.99	65.29	39.15	40.02
File8	25210	14412	14488	6626	9865	57.37	57.68	26.28	39.13
File9	42080	19359	19688	10256	11698	46.01	46.79	24.37	27.80
File10	50464	22430	22864	17440	14493	44.44	45.31	34.56	28.72
File11	60456	38003	38264	19769	19847	62.86	63.29	32.70	32.83
Avg.	**16897.82**	**8979.55**	**9081.73**	**5156.91**	**5397.64**	**53.06**	**53.45**	**31.03**	**44.47**