Volume 4, Issue 7, July 2014 ISSN: 2277 128X

International Journal of Advanced Research in Computer Science and Software Engineering

Research Paper

Available online at: www.ijarcsse.com

An Improved Dynamic Bit Reduction Algorithm for Lossless Text Data Compression

Neha Sharma
M. Tech Scholar
Computer Science & Engineering
GZS PTU Campus,
Bathinda, India

Dr. Paramjeet Singh
Assistant Professor
Computer Science & Engineering
GZS PTU Campus,
Bathinda, India

Dr. Shaveta Rani
Assistant Professor
Computer Science & Engineering
GZS PTU Campus,
Bathinda, India

Abstract— Data compression is a very useful technique that helps in reducing the size of text data and storing the same amount of data in relatively fewer bits resulting in reducing the required data storage space. Text data compression techniques can be categorized as Lossy and Lossless. In this paper, we propose an Improved Dynamic Bit Reduction Algorithm for compression and decompression of text data based on lossless data compression approach. Improved Dynamic bit reduction algorithm identifies the no. of unique characters in the input string and performs the compression process by converting the ASCII characters in to binary code dynamically based upon the occurrence of unique symbols in the input string. Huffman coding algorithm is used to optimize the performance of dynamic bit reduction. The performance of the proposed algorithm is measured and compared with the existing systems using parameters- Compression Ratio and Saving Percentage.

Keywords— Compression ratio, Dynamic Bit reduction, Huffman coding, Lossless compression, Saving Percentage, Text Data Compression

I. INTRODUCTION

Data Compression is the process of encoding data to fewer bits so that it takes less storage space or less transmission time that it would if it were not compressed [1].

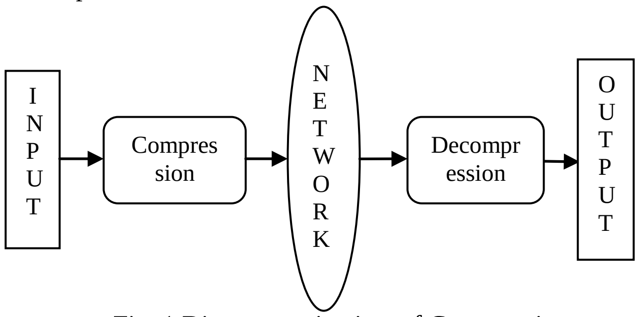


Fig. 1 Diagrammatic view of Compression

Compression is possible because most of the real world data is very redundant. Data Compression is a technique that reduces the size of data by removing unnecessary information and duplicity. Data compression is an art of reducing the number of bits needed to store or transmit data.

Mainly, two types of data compression techniques are there: **Lossy and Lossless** data compression [1].

A **lossy data compression method** is one where compressing data and then decompressing it retrieves data that may not be exactly same as the original, but is "close enough" to be useful for specific purpose. The examples of frequent use of Lossy data compression are on the Internet and especially in the streaming media and telephony applications. Most of the lossy data compression techniques suffer from generation loss which means decreasing the quality of text because of repeatedly compressing and decompressing the file. Lossy image compression can be used in digital cameras to increase storage capacities with minimal degradation of picture quality.

Lossless data compression is a technique that allows the use of data compression algorithms to compress the data and also allows the exact original data to be reconstructed from the compressed data. This is in contrary to the lossy data compression in which the exact original data cannot be reconstructed from the compressed data. The popular ZIP file format that is being used for the compression of data files is also an application of lossless data compression approach. Lossless compression is used when it is important that the original data and the decompressed data be identical. Lossless text data compression algorithms usually exploit statistical redundancy in such a way so as to represent the sender's data more concisely without any error or any sort of loss of important information contained within the text input data. Since most of the real-world data has statistical redundancy, therefore lossless data compression is possible. For instance, In English text, the letter 'a' is much more common than the letter 'z', and the probability that the letter't' will be followed by the letter 'z' is very small. So this type of redundancy can be removed using lossless compression. Lossless compression methods may be categorized according to the type of data they are designed to compress. Compression algorithms are basically used for the compression of text, images and sound. Most lossless compression programs use two different kinds of algorithms: one which generates a statistical model for the input data and another which maps the input data to bit strings using this model in such a way that frequently encountered data will produce shorter output than improbable(less frequent) data. The advantage of lossless methods over lossy methods is that Lossless compression results are in a closer representation of the original input data. The performance of the algorithms can be measured using the parameters such as Compression Ratio and Saving Percentage.

Compression Ratio: Compression ratio is defined as the ratio of size of the compressed file to the size of the source file [2].

Compression ratio=C2/C1 *100%

Saving Percentage: Saving Percentage calculates the shrinkage of the source file as a percentage.

Saving percentage =(C1-C2/C1)*100%

C1=Size before compression

C2=Size after compression

II. EXISTING WORK

A. Bit Reduction algorithm:

The main idea behind this program is to reduce the standard 7-bit encoding to some application specific 5-bit encoding system and then pack into a byte array. This algorithm was originally implemented to use in a text file like message communication application [3].

Bit Reduction Algorithm in steps-

- Select the frequently occurring characters from the text file which are to be encoded and obtain their corresponding ASCII code.
- Obtain the corresponding binary code of these ASCII key codes for each character.
- Then put these binary numbers into an array of byte (8bit array).
- Remove extra bits from binary no like extra 3 bits from the front.
- Then rearrange these into array of byte and maintain the array.
- Final text will be encoded and as well as compression will be achieved.
- Now decompression will be achieved in reverse order at the client-side.

B. Huffman Coding:

Huffman coding deals with data compression of ASCII characters. It follows top down approach means the binary tree is built from the top down to generate an optimal result. In Huffman Coding the characters in a data file are converted to binary code and the most common characters in the file have the shortest binary codes, and the characters which are least common have the longest binary code [4]. The algorithm to generate Huffman code is:

1. Parse the input and count the occurrence of each symbol.
2. Determine the probability of occurrence of each symbol using the symbol count.
3. Sort the symbols according to their probability of occurrence, with the most probable first.
4. Then generate leaf nodes for each symbol and add them to a queue.
5. Take two least frequent characters and then logically group them together to obtain their combined frequency that leads to the construction of a binary tree structure.
6. Repeat step 5 until all elements are reached and there remains only one parent for all nodes which is known as root.
7. Then label the edges from each parent to its left child with the digit 0 and the edge to right child with 1. Tracing down the tree yields to “Huffman codes” in which shortest codes are assigned to the character with greater frequency.

III. PROPOSED SYSTEM

Proposed System works in two phases to compress the text data. In the first phase data is compressed with the help of dynamic bit reduction technique and in second phase Huffman coding is used to compress the data further to produce the final output. In the similar way method of decompression works in reverse order. Compressed data is first decompressed by Huffman Decoder and then by dynamic bit reduction decoder. Following are the steps to compress the data with the help of our proposed system:

Steps for Data Compression:

Step I: Input the text data to be compressed.

Step II: Find the number of unique symbols in the input text data.

Step III: Assign the numeric code to the unique symbols found in the step II.

Step IV: Starting from first symbol in the input find the binary code corresponding to that symbols from assigned numerical codes and concatenate them to obtain binary output.

Step V: Add number of 0’s in MSB of Binary output until it is divisible by 8.

Step VI: Generate the ASCII code for every 8 bits for the binary output obtained in step V and concatenate them to create input for second phase.
[Step VI is the result of dynamic bit Reduction Method]
Step VII: Give the output generated by Step VI to Huffman tree to further compress the data and obtain the result.
Step VIII: Display the final result obtained in step VII.
[Output from step VIII is final compressed output]

Steps for Decompression
Step I: Input the Final output from compressed phase.
Step II: Assign this input to the Huffman decoder to decompress the data compressed by Huffman tree in ASCII format.
Step III: Calculate the binary code corresponding to the ASCII values obtained in Step II.
Step IV: Remove the extra bits from the binary output added in the compression phase.
Step V: Calculate the numeric code for every 8 bits obtained in the Step IV.
Step VI: For every numeric value obtained in the step V, find the corresponding symbol to get the final decompressed data.
Step VII: Concatenate the data symbols obtained in the step VI and obtain the final output.
Step VIII: Display the final result to the user.

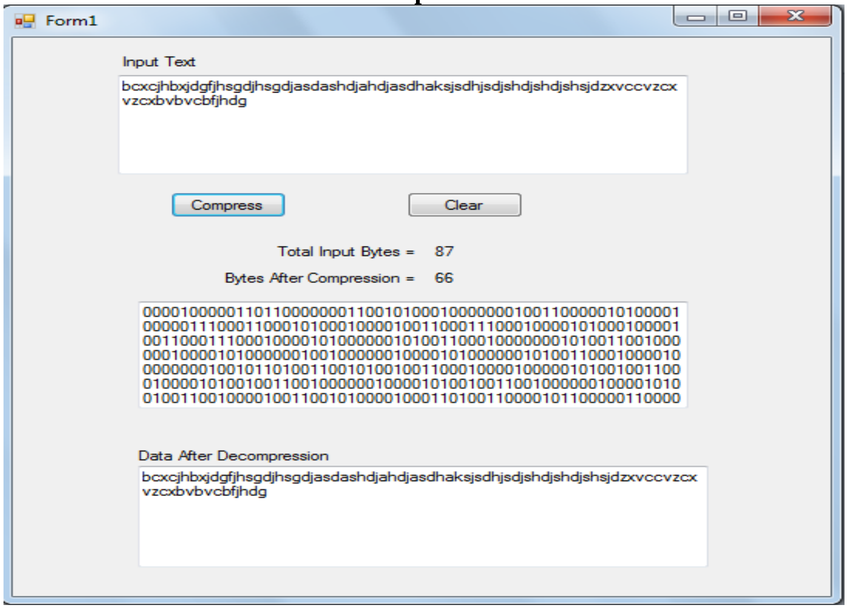
IV. RESULTS AND DISCUSSION

This section involves the implementation of existing and the proposed system and analysing their performance in terms of Compression Ratio.

A. Bit Reduction Output

Implementation: This algorithm has been implemented using C# programming language and VS 2008 as an Integrated Development Environment.

Snapshot



Input Text
bcxcjhbxjdgfjhsgdjhsgdjasdashdjahdjasdhaksjsdhjsdjshdjshdjshsjdzxvccvzcx
vzcxbvbvcbfjhdg

Total Input Bytes = 87
Bytes After Compression = 66

```
0000100000110110000000011001010001000000001001100000010100001
0000001110001100010100010000100110001110001000010100010000 1
0011000111000100001010000000101001100010000000001010011001000
0001000010100000001001000000100001010000000101001100010000010
0000000100101101001100101001001100010000100000101001001100
0100001010010011001000000010000101001001100100000001000001010
0100110010000100110010100001000110100110000101100000110000
```

Data After Decompression
bcxcjhbxjdgfjhsgdjhsgdjasdashdjahdjasdhaksjsdhjsdjshdjshdjshsjdzxvccvzcx
vzcxbvbvcbfjhdg

Fig. 2 Output of Bit reduction algorithm

TABLE I. BIT REDUCTION OUTPUT

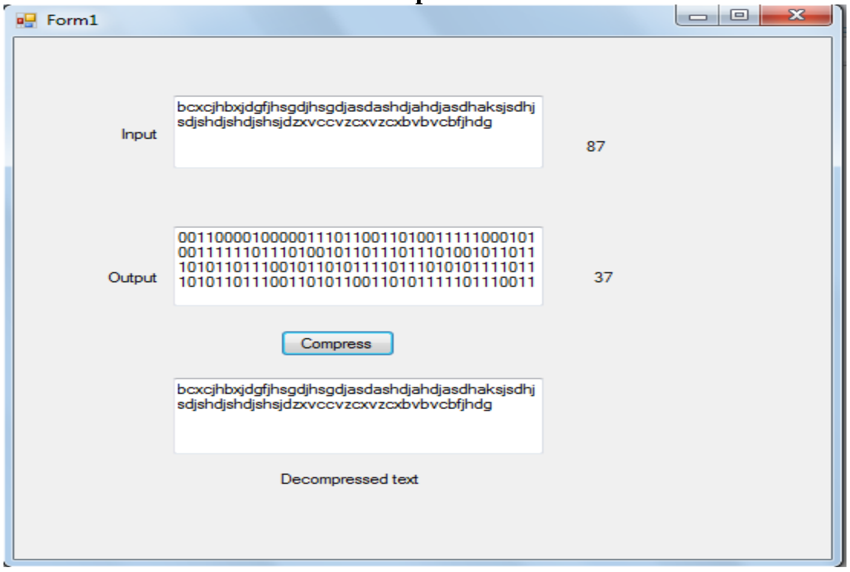
Input text size (in bytes)	Compressed Output (in bytes)	Saving percentage	Compression ratio (in %)
87	66	24.1	75.8
117	88	24.7	75.2
176	132	25	75
352	264	25	75
536	402	25	75

Table I shows the various experiments conducted by the authors to determine the compression ratio achieved by bit reduction algorithm for text data of varying lengths. The compression ratio of bit reduction algorithm lies around 75% which means that it saves around 25% of the storage space in memory.

B. Huffman coding Output

Implementation: This algorithm has also been implemented using C# programming language and VS 2008 as an Integrated Development Environment.

Snapshot



```
Input
bcxcjhbxjdgfjhsgdjhsgdjasdashdjahdjasdhaksjsdhj
sdjshdjshdjshsjdzxvccvzcxvzcxbvbvcbfjhdg
87

Output
0011000010000001110110011010011111000101
0011111101110100101101110111010010111011
101011011100101101011110111010101111011
1010110111001101011001101011111011100111
37

bcxcjhbxjdgfjhsgdjhsgdjasdashdjahdjasdhaksjsdhj
sdjshdjshdjshsjdzxvccvzcxvzcxbvbvcbfjhdg
```

Fig. 3 Output of Huffman coding algorithm

TABLE II. HUFFMAN CODING OUTPUT

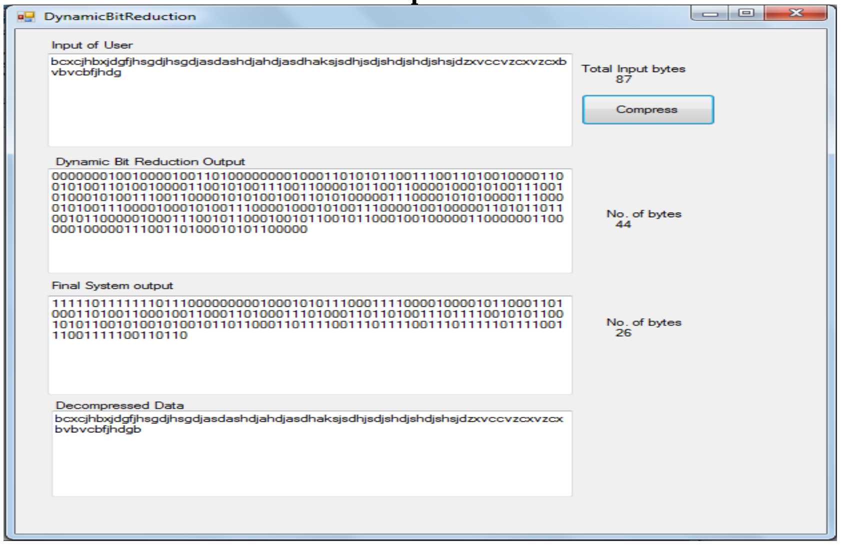
Input text size (in bytes)	Output (in bytes)	Saving percentage	Compression ratio (in %)
87	37	57.4	42.5
117	50	57.2	42.7
176	102	42	57.9
352	204	42	57.9
536	310	42.1	57.8

Table II shows the various experiments conducted by the authors to determine the compression ratio achieved by Huffman coding algorithm for text data of varying lengths.

C. Proposed System Output

Implementation: This algorithm has also been implemented using C# programming language and VS 2008 as an Integrated Development Environment

Snapshot



```
Input of User
bcxcjhbxjdgfjhsgdjhsgdjasdashdjahdjasdhaksjsdhjsdjshdjshdjshsjdzxvccvzcxvzcxb
vbvcbfjhdg
Total Input bytes
87

Dynamic Bit Reduction Output
00000001001000010011010000000001000110101011001110011010010000110
01010011010010000110010100111001100001011001100001000101001 11001
01000101001110011000010101001001101010000011100001010100001 11000
01010011100001000101001110000100010100111000010010000011010110 11
0010110000001000111001011000100101100101100010010000011000000 1100
0001000000111001101000101011 00000
No. of bytes
44

Final System output
1111101111111011100000000001000101011100011110000100001011000110 1
00011010011000100110001101000111010001101101001110111100101011 00
1010110010100101001011011000110111100111011110011101111110111100 1
1100111110011011 0
No. of bytes
26

Decompressed Data
bcxcjhbxjdgfjhsgdjhsgdjasdashdjahdjasdhaksjsdhjsdjshdjshdjshsjdzxvccvzcxvzcx
bvbvcbfjhdgb
```

Fig.4 Output of Proposed System

TABLE III. PROPOSED SYSTEM OUTPUT

Input text size (in bytes)	**Output of proposed System (in bytes)**	**Saving Percentage**	**Compression Ratio of proposed system (In %)**
87	26	70.1	29.8
117	36	69.2	30.7
176	87	50.5	49.4
352	174	50.5	49.4
536	286	46.6	53.3

Table III shows the various experiments conducted by the authors to determine the compression ratio achieved by the final proposed system. The compression ratio of the proposed system lies between 29-53% which means that it saves around 47-71% of the storage space in memory which is better than the existing techniques.

D. Comparison Tables and Graphs

The following table and graph represents the comparison of Compression ratios of the existing techniques and the proposed system.

TABLE IV. COMPRESSION RATIO COMPARISON

Input text size (in bytes)	**Bit Reduction Compression ratio (In %)**	**Huffman Compression ratio (In %)**	**Proposed System Compression ratio (in %)**
87	75.8	42.5	29.8
117	75.2	42.7	30.7
176	75	57.9	49.4
352	75	57.9	49.4
536	75	57.8	53.3

From Table IV, it is clear that the compression ratio achieved by the proposed system is lesser as compared to the existing techniques which means it results in more savings of the storage space.

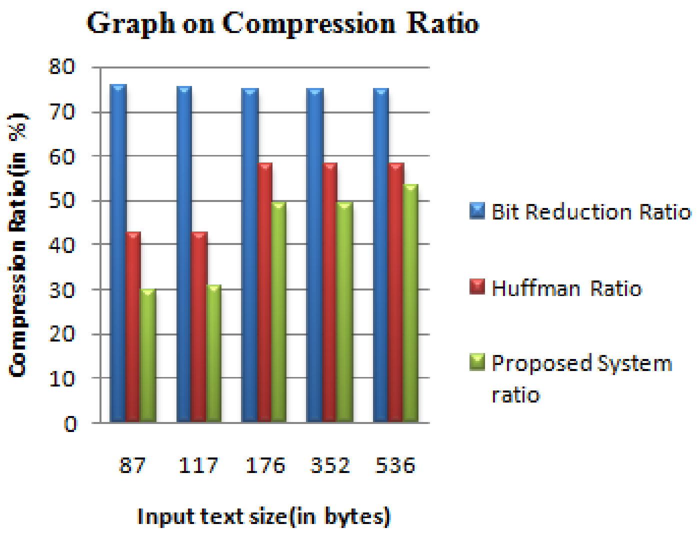


Fig.5 Graph showing the comparison of Compression ratios

The above graph (Fig. 5) is made on the basis of Table IV which shows the comparison of compression ratios of the existing systems and the proposed system. In the graph, the horizontal axis represents the length of input string in bytes and vertical axis represents the Compression Ratio in percentage.

The following table and graph represents the comparison of Saving Percentage of the existing techniques and the proposed system:

TABLE V COMPARISON OF SAVING PERCENTAGE

Input text size (in bytes)	Saving Percentage of Bit Reduction	Saving Percentage of Huffman coding	Saving Percentage of Proposed System
87	24.1	57.4	70.1
117	24.7	57.2	69.2
176	25	42	50.5
352	25	42	50.5
536	25	42.1	46.6

From Table V, it is clear that the saving percentage of proposed system is higher as compared to the existing techniques.

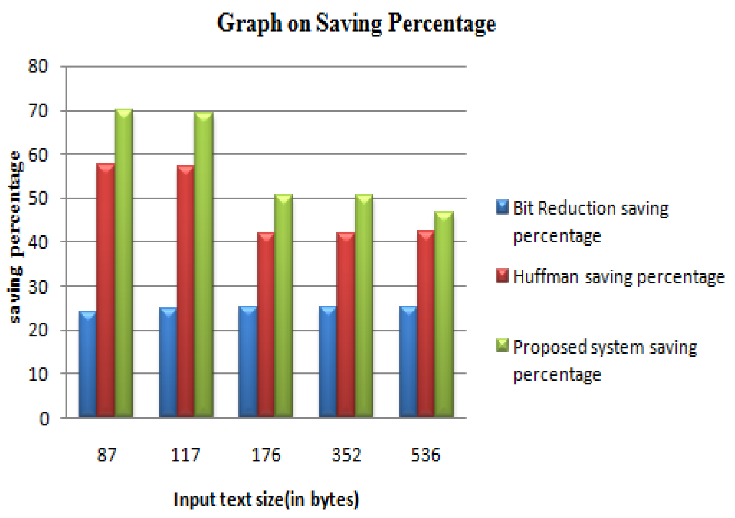


Fig.6 Graph showing the comparison of saving percentage

The above graph (Fig. 6) is made on the basis of Table V which shows the comparison of saving percentage of existing systems and the proposed system. In the graph, the horizontal axis represents the length of input text string in bytes and vertical axis represents the saving percentage.

V. CONCLUSION

In this paper, an Improved Dynamic Bit Reduction Algorithm to compress and decompress the text data has been introduced. We also present the results obtained by the proposed system and compare these results with the existing data compression techniques. The limitation of the existing bit reduction algorithm is that it does not provide good compression results and provides lossy output if special characters are present in the input string. These limitations have been overcome by the proposed system. Proposed system shows very good compression results in terms of compression ratio and saving percentage and proved better than the existing techniques for these parameters. Improved dynamic bit reduction algorithm works only with text data which can also be tested to compress the multi lingual data for future work.

ACKNOWLEDGEMENT

I would like to thank Dr. Paramjeet Singh, Assistant Professor (CSE), Giani Zail Singh Punjab Technical University Campus, Bathinda and Dr. Shaveta Rani for their valuable suggestions, constant support and guidance. I would also like to express my gratitude towards the Computer Science Department of GZS PTU Campus, Bathinda, India for providing the resources to proceed with the research work.

REFERENCES

[1] R.S. Brar and B.Singh, “A survey on different compression techniques and bit reduction algorithm for compression of text data” *International Journal of Advanced Research in Computer Science and Software Engineering (IJARCSSE)* Volume 3, Issue 3, March 2013

[2] S. Porwal, Y. Chaudhary, J. Joshi, M. Jain, “Data Compression Methodologies for Lossless Data and Comparison between Algorithms” *International Journal of Engineering Science and Innovative Technology (IJESIT)* Volume 2, Issue 2, March 2013

[3] R.Kaur and M.Goyal, “An Algorithm for Lossless Text Data Compression” *International Journal of Engineering Research & Technology (IJERT),* Vol. 2 Issue 7, July - 2013

[4] S. Shanmugasundaram and R. Lourdusamy, "A Comparative Study of Text Compression Algorithms" *International Journal of Wisdom Based Computing*, Vol. 1 (3), December 2011

[5] H. Altarawneh and M. Altarawneh,"Data Compression Techniques on Text Files: A Comparison Study" *International Journal of Computer Applications*, Volume 26– No.5, July 2011

[6] U .Khurana and A.Koul, "Text Compression And Superfast Searching", *Thapar Institute Of Engineering and Technology*, Patiala, Punjab, India-147004

[7] A. Singh and Y. Bhatnagar, "Enhancement of data compression using Incremental Encoding" *International Journal of Scientific & Engineering Research*, Volume 3, Issue 5, May-2012

[8] M.Sharma, "Compression using Huffman Coding" *IJCSNS International Journal of Computer Science and Network Security*, VOL.10 No.5, May 2010

[9] A.J Mann, "Analysis and Comparison of Algorithms for Lossless Data Compression" *International Journal of Information and Computation Technology*, ISSN 0974-2239 Volume 3, Number 3 (2013), pp. 139-146

[10] P.Yellamma and Dr.N.Challa," Performance Analysis of Different Data Compression Techniques on Text File" *International Journal of Engineering Research & Technology (IJERT)* Vol. 1 Issue 8, October – 2012 ISSN: 2278-0181

[11] J.M. Jou and P.Y. Chen, "A Fast and Efficient Lossless Data Compression Method" *IEEE TRANSACTIONS ON COMMUNICATIONS*, VOL. 47, NO. 9, SEPTEMBER 1999

[12] H.J. Kim, "A NEW LOSSLESS DATA COMPRESSION METHOD" *978-1-4244-4291-1/09/$25.00 ©2009 IEEE*

[13] K.Rastogi, K.Sengar, "Analysis and Performance Comparison of Lossless Compression Techniques for Text Data" *International Journal of Engineering Technology and Computer Research (IJETCR)* 2 (1) 2014, 16-19

[14] P.Kumar and A.K Varshney, "Double Huffman Coding" *International Journal of Advanced Research in Computer Science and Software Engineering (IJARCSSE)"* Volume 2, Issue 8, August 2012

[15] A.D Suarjaya, "A New Algorithm for Data Compression Optimization" *International Journal of Advanced Computer Science and Applications (IJACSA)"* Vol. 3, No.8, 2012