RESEARCH ARTICLE OPEN ACCESS

An Enhanced Text Compression System Based on ASCII Values and Huffman Coding

Mamta Rani [1], Vikram Singh [2]
Department of Computer Science and Applications
Chaudhary Devi Lal University, Sirsa
Haryana, India

ABSTRACT

In today's environment a large amount of data is required to be sent from one place to another place for communication or some other purpose. A large amount of data communication results into high cost and lower down the performance. Hence researchers have developed various systems to compress the text data to reduce data transfer cost and to increase the performance of the communication channel. Basically Data compression is a process by which a file (Text, Audio, and Video) may be transformed to another (compressed) file, such that the original file may be fully recovered from the original file without any loss of actual information. This process may be useful if one wants to save the storage space. Also compressed files are much more easily exchanged over the internet since they can be uploaded and downloaded much faster. In the present paper a data compression algorithm is represented using dynamic text compression algorithm which uses Huffman Coding to compress and decompress the text data. This algorithm improves the performance of Huffman Algorithm and gives better results.

Keywords:- Text Data Compression, Huffman Coding, Enhanced Dynamic Text Compression System, Lossless Data Compression.

I. INTRODUCTION

Data compression is a process by which a file (Text, Audio, Video) may be transformed to another (compressed) file, such that the original file may be fully recovered from the original file without any loss of actual information. This process may be useful if one wants to save the storage space. Data compression is a process that reduces the data size, removing the excessive information and redundancy. Why shorter data sequence is more suitable? –the answer is simple it reduces the cost. Data compression is a common requirement for most of the computerized application. Data compression has important application in the area of file storage and distributed system. Data compression is used in multimedia field, text documents, and database table.

A. Types of Data Compression

Data compression methods can be classified in several ways. One of the most important criteria of classification is whether the compression algorithms remove some part of data which cannot be recovered during decompression.

1) ***Lossy Data Compression***: The algorithm which removes some part of data is called lossy data compression. The lossy data compression algorithm is usually used when a perfect consistency with the original data is not necessary after decompression. Example of lossy data compression is compression of video or picture data.
2) ***Lossless Data Compression:*** The algorithm that achieve the same what we compressed after decompression is called lossless data compression as in [1]. Lossless data compression is used in text file, database tables and in medical image because law of regulations. Various lossless data compression algorithm have been proposed and used. Some of main techniques are Huffman Coding, Run Length Encoding, Arithmetic Encoding and Dictionary Based Encoding.

B. EXISTING LOSSLESS DATA COMPRESSION TECHNIQUES

The existing lossless data compression techniques are described as follow:

1) ***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 method reduces the size of a string considerably when the string is lengthy and the compression ratio is not affected by the content of the string.
2) ***Huffman Coding:*** Huffman Encoding Algorithms use the probability distribution of the alphabet of the source

to develop the code words for symbols. The frequency distribution of all the characters of the source is calculated in order to calculate the probability distribution. According to the probabilities, the code words are assigned. Shorter code words for higher probabilities and longer code words for smaller probabilities are assigned. For this task a binary tree is created using the symbols as leaves according to their probabilities and paths of those are taken as the code words. Two families of Huffman Encoding have been proposed: Static Huffman Algorithms and Adaptive Huffman Algorithms. Static Huffman Algorithms calculate the frequencies first and then generate a common tree for both the compression and decompression processes as in [2]. Details of this tree should be saved or transferred with the compressed file. The Adaptive Huffman algorithms develop the tree while calculating the frequencies and there will be two trees in both the processes. In this approach, a tree is generated with the flag symbol in the beginning and is updated as the next symbol is read.

3) ***Run length Encoding:*** Run Length Encoding or simply RLE is the simplest of the data compression algorithms. The consecutive sequences of symbols are identified as runs and the others are identified as non runs in this algorithm. This algorithm deals with some sort of redundancy as in [2]. It checks whether there are any repeating symbols or not, and is based on those redundancies and their lengths. Consecutive recurrent symbols are identified as runs and all the other sequences are considered as non-runs. For an example, the text "ABABBBBC" is considered as a source to compress, then the first 3 letters are considered as a non-run with length 3, and the next 4 letters are considered as a run with length 4 since there is a repetition of symbol B. The major task of this algorithm is to identify the runs of the source file, and to record the symbol and the length of each run.

4) ***Shannon-Fano coding:*** This is one of an earliest technique for data compression that was invented by Claude Shannon and Robert Fano as in [3] in 1949.In this technique, a binary tree is generated that represent the probabilities of each symbol occurring. The symbols are ordered in a way such that the most frequent symbols appear at the top of the tree and the least likely symbols appear at the bottom . Generally, Shannon-Fano coding does not guarantee the generation of an optimal code. Shannon – Fano algorithm is more efficient when the probabilities are closer to inverses of powers of 2 .

5) ***Arithmetic Coding:*** Arithmetic coding is an optimal entropy coding technique as it provides best compression ratio and usually achieves better results than Huffman Coding. It is quite complicated as compared to the other coding techniques. When a string is converted in to arithmetic encoding, the characters having maximum probability of occurrence will be stored with fewer bits and the characters that do not occur so frequently will be stored with more bits, resulting in fewer bits used overall. Arithmetic coding converts the stream of input symbols into a single floating point number as output as in [3]. Unlike Huffman coding, arithmetic coding does not code each symbol separately. Each symbol is instead coded by considering all prior data. Thus a data stream encoded in this fashion must always be read from the beginning.

6) ***Lempel-Ziv-Welch (LZW) Algorithm****:* Dictionary based compression algorithms are based on a dictionary instead of a statistical model. A dictionary is a set of possible words of a language, and is stored in a table like structure and used the indexes of entries to represent larger and repeating dictionary words. The Lempel-Ziv Welch algorithm or simply LZW algorithm is one of such algorithms. In this method, a dictionary is used to store and index the previously seen string patterns. In the compression process, those index values are used instead of repeating string patterns. The dictionary is created dynamically in the compression process and no need to transfer it with the encoded message for decompressing. In the decompression process, the same dictionary is created dynamically. Therefore, this algorithm is an adaptive compression algorithm.

II. LITERATURE REVIEW

In (A. Jain, R. Patel,2009) as in [4] authors presented an intelligent, reversible transformation technique that can be applied to source text that improve algorithm ability to compress and also offer a sufficient level of security to the transmitted data. In this paper, the authors present an encoding technique (ECA) which offers highest compression ratios. The authors suggest that in an ideal channel, the reduction of transmission time is directly proportional to the amount of compression. But in a typical Internet scenario with fluctuating bandwidth, congestion and protocols of packet switching, this does not hold true. The

authors conclude that their results have shown excellent improvement in text data compression and added levels of security over the existing methods. These improvements come with additional processing required on the server/node.
In (K. Rastogi and K. Sengar, 2014) as in [5] the author discussed about the lossless text data compression algorithms such as Run Length Encoding, Huffman Coding and Shanon Fano coding. The authors have concluded the article by doing a comparison of these techniques. The authors have also concluded that Huffman technique is most optimal for lossless data compression.

M. Sharma(2010) as in [6] has analyzed Huffman algorithm and compare it with other common compression techniques like Arithmetic, Lempel Ziv Welch(LZW) and Run Length Encoding(RLE). The author has concluded that arithmetic coding is very efficient for bits and reduces the file size dramatically.RLE is simple to implement and fast o execute. LZW algorithm is better to use for TIFF, GIF and Textual Files.

In (P. Kumar A.K. Varshney,2012) as in [7] the authors presented a novel technique that work on Huffman Coding and after getting codeword for the Symbol the authors compress it on the Basis of its Binary no. 0 and 1. The authors have analyzed the results by comparing Double Huffman coding with Huffman coding and concluded that the Double Huffman coding despite being costly provides better results in terms of performance and space saving.

III. PROPOSED SYSTEM

To improve the compression ratio and memory saving percentage for text data, a lossless compression algorithm named as "Enhanced Dynamic Text Compression Algorithm" is designed which is specialized for text data. The basic idea of the devised compression algorithm is the introduction of a new encoding scheme which also uses Huffman coding technique for all redundant words in text files to improve the performance of the proposed system. The algorithm has been implemented on C# platform with Visual Studio 2008 as an IDE (Integrated Development Environment).

A. Proposed System Methodology

Enhanced Dynamic Text Compression algorithm 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 First phase, when user enters an input text data, the system will find out the occurrence of number of unique symbols in the input text string and then numeric code will be assigned to these unique symbols. For each numeric code, corresponding binary codes will be generated dynamically to obtain the (compressed) binary output. Then ASCII code will be generated from the binary output obtained which will serve as the input to the second phase of the system. In the second phase Huffman Coding will be applied to the output of first phase to further compress the data and improve the performance of dynamic bit reduction algorithm. Huffman coding follows top down approach means the binary tree is built from the top to 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 will have the longest binary code. In the similar way method of decompression works in reverse order. Compressed data is first decompressed by Huffman Decoder and then by dynamic text compression decoder to get back the original data. Following are the steps to compress the data with the help of our proposed system.

B. Compression Algorithm

1. Input the text data to be compressed.
2. Find the number of unique symbols in the input text data.
3. Assign the numeric code to the unique symbols found in the step 2.
4. 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.
5. Add number of 0's in MSB of Binary output until it is divisible by 8.
6. Generate the ASCII code for every 8 bits for the binary output obtained in step 5 and concatenate them to create input for second phase. [Step 6 is the result of dynamic text compression Method in ASCII format]
7. Give the output generated by Step 6 to Huffman tree to further compress the data and obtain the result in compressed binary output form.
8. Display the final result obtained in step 7.

[Output from step 8 is final compressed output]

The algorithm to generate Huffman Tree 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, keeping 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.

C. Decompression Algorithm

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

D. Performance Evaluation Parameters

Performance evaluation of the proposed algorithm is done using two parameters-Compression Ratio and Saving Percentage.

1) *Compression ratio:* Compression ratio is defined as the ratio of size of the compressed file to the size of the source file.

$$CompressionRatio = (C2/C1)$$

where C1= Size before compression

C2= Size after compression

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

$$SavingPercentage = ((C1-C2)/C1)*100\%$$

where C1= Size before compression

C2= Size after compression

IV. RESULTS AND DISCUSSION

A. Comparison Table and Graph on Compression Ratio for Random Dataset: The following tables and graphs represent the comparison of Compression ratios of the existing techniques and the proposed system.

Input text size (in bytes)	Huffman Output Text Size	Huffman Compression ratio	Proposed System Output Text Size	Proposed System Compression ratio
36	13	36.11	6	16.67
55	26	47.27	15	27.27
76	33	43.42	17	22.37
140	79	56.42	67	47.85
227	123	54.19	79	34.80

TABLE I COMPRESSION RATIO COMPARISON FOR RANDOM DATA SET

From Table I, 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.

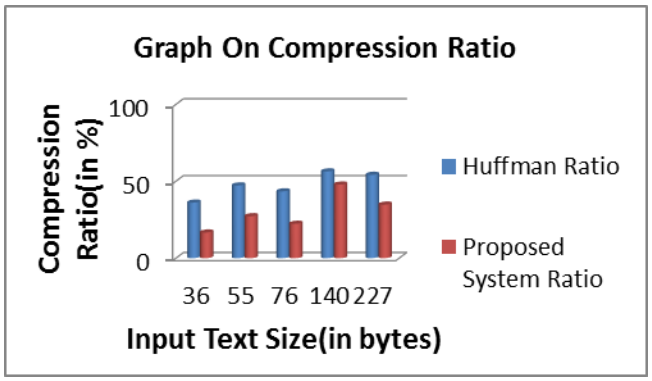

Fig.1 Compression ratio comparison graph for random dataset

The above graph is made on the basis of Table I 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.

*B) **Comparison Table and Graph on Saving Percentage for Random Dataset:***

The following tables and graphs represent the comparison of saving percentage of the existing techniques and the proposed system.

Input text size (in bytes)	Huffman saving percentage	Proposed System saving percentage
36	63.88	83.33
55	52.72	72.73
76	56.58	77.63
140	43.57	52.14
227	45.81	65.20

TABLE II SAVING PERCENTAGE COMPARISON FOR RANDOM DATASET

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

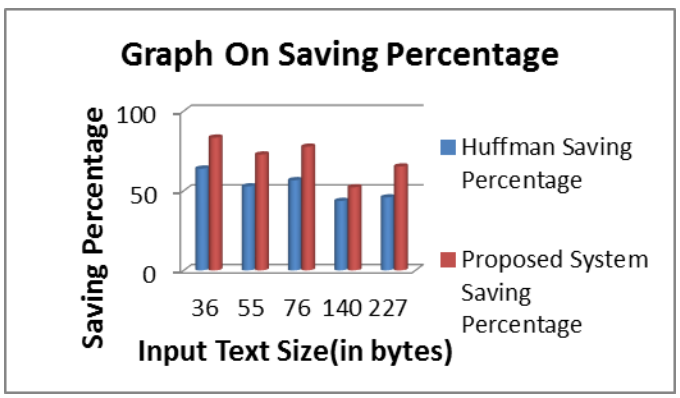

Fig. 2 Saving percentage comparison graph for random dataset

The above graph is made on the basis of Table II 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 enhanced dynamic text compression algorithm is represented to compress and decompress the text data based on lossless data compression approach. Various experiments have been conducted on different datasets such as Random, Alphanumeric, Numeral and Special Characters dataset.

From the results analysis, it is concluded that the proposed system shows very good compression results in terms of Compression Ratio and Saving Percentage as compared to the existing techniques for all the datasets that have been considered.

Enhanced Dynamic Text Compression Algorithm works only with text data written in single language which can also be tested to compress the multi lingual data i.e. text data written in multiple languages in a single file.

REFERENCES

[1] Shrusti Porwal, Yashi Chaudhary, Jitendra Joshi, Manish 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.

[2] Blelloch, E., 2002. Introduction to Data Compression, Computer Science Department, Carnegie Mellon University.

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

[4] A. Jain, R. Patel, " An Efficient Compression Algorithm (ECA) for Text Data "International Conference on Signal Processing Systems, 2009 IEEE.

[5] [5] 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.

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

[7] 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.