Beyond syllabus

AIM : study of a research paper focusing on latest innovation in Data Compression

1. Introduction

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. For example, if one wants to store a 4MB file, it may be preferable to first compress it to a smaller size to save the storage space.

Also compressed files are much more easily exchanged over the internet since they upload and download much faster. We require the ability to reconstitute the original file from the compressed version at any time. Data compression is a method of encoding rules that allows substantial reduction in the total number of bits to store or transmit a file. The more information being dealt with, the more it costs in terms of storage and transmission costs. In short, Data Compression is the process of encoding data to fewer bits than the original representation so that it takes less storage space and less transmission time while communicating over a network [1].

Data Compression is possible because most of the real-world data is very redundant. Data Compression is basically defined as a technique that reduces the size of data by applying different methods that can either be Lossy or Lossless [1]. A compression program is used to convert data from an easy-to-use format to one optimized for compactness. Likewise, an uncompressing program returns the information to its original form.

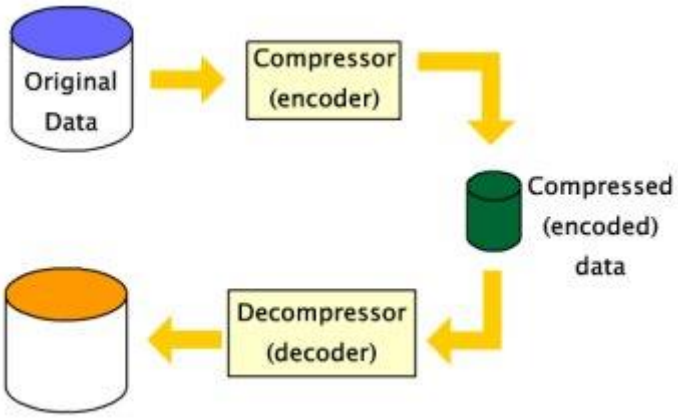

Fig. 1.1 Data Compression and Decompression

1.2 TYPES OF DATA COMPRESSION

Currently, two basic classes of data compression are applied in different areas. One of these is lossy data compression, which is widely used to compress image data files for communication or archives purposes. The other is lossless data compression that is commonly used to transmit, or archive text or binary files required to keep their information intact at any time.

There are two mainly two types of Data Compression:

- Lossy Compression
- Lossless Compression

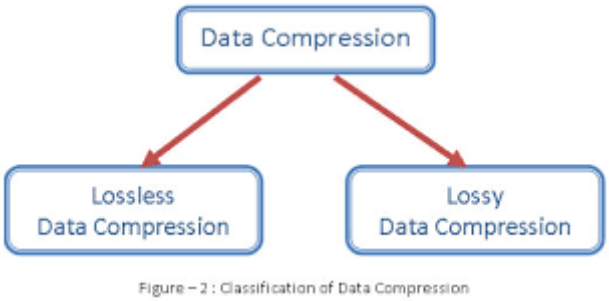

Figure – 2 : Classification of Data Compression

Fig: 1.2 Classification of Data Compression

1.2.1 Lossy Data Compression

A lossy data compression method is one where the data retrieves after decompression may not be exactly same as the original data but is "close enough" to be useful for specific purpose. After one applies lossy data compression to a message, the message can never be recovered exactly as it was before it was compressed. When the compressed message is decoded it does not give back the original message. Data has been lost. Because lossy compression cannot be decoded to yield the exact original message, it is not a good method of compression for critical data, such as textual data. It is most useful for Digitally Sampled Analog Data (DSAD). DSAD consists mostly of sound, video, graphics, or picture files. In a sound file, for example, the very high and low frequencies, which the human ear cannot hear, may be truncated from the file.

The examples of frequent use of Lossy data compression are on the Internet and especially in the streaming media and telephony applications. Some examples of lossy data compression algorithms are JPEG, MPEG, MP3.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.

1.2.2 Lossless Data Compression

Lossless data compression is a technique that allows the use of data compression algorithms to compress the text 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 algorithms can be compared using the parameters such as Compression Ratio and Saving Percentage. In a lossless data compression file the original message can be exactly decoded. Lossless data compression works by finding repeated patterns in a message and encoding those patterns in an efficient manner. For this reason, lossless data compression is also referred to as redundancy reduction. Because redundancy reduction is dependent on patterns in the message, it does not work well on random messages. Lossless data compression is ideal for text.

2. LITERATURE REVIEW

This section involves the Literature survey of various techniques available for Data compression and analyzing their results and conclusions.

R. S. Brar and B. Singh, et.al, (2013), “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.This paper provides a survey of different basic lossless and lossy data compression techniques. On the basis of these techniques a bit reduction algorithm for compression of text data has been proposed by the authors based on number theory system and file differential technique which is a simple compression and decompression technique free from time complexity. Future work can be done on coding of special characters which are not specified on key-board to revise better results. S. Porwal, Y. Chaudhary, J. Joshi, M. Jain, et. al, (2013), “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. This research paper provides lossless data compression methodologies and compares their performance. Huffman and arithmetic coding are compared according to their performances. In this paper the author has found that arithmetic encoding methodology is powerful as compared to Huffman encoding methodology. By comparing the two techniques the author has concluded that the compression ratio of arithmetic encoding is better and furthermore arithmetic encoding reduces channel bandwidth and transmission time also.

S.Shanmugasundaram and R. Lourdusamy, et. al, (2011), “A Comparative Study of Text Compression Algorithms”. International Journal of Wisdom Based Computing, Vol.1 (3), Dec 2011. There are lot of data compression algorithms which are available to compress files of different formats. This paper provides a survey of different basic lossless data compression algorithms. Experimental results and comparisons of the lossless compression algorithms using Statistical compression techniques and Dictionary based compression techniques were performed on text data. Among the statistical coding techniques, the algorithms such as Shannon-Fano Coding, Huffman coding, Adaptive Huffman coding, Run Length Encoding and Arithmetic coding are considered. A set of interesting conclusions are derived on their basis. Lossy algorithms achieve better compression effectiveness than lossless algorithms, but lossy compression is limited to audio, images, and video, where some loss is

acceptable. The question of the better technique of the two, “lossless” or “lossy” is pointless as each has its own uses with lossless techniques better in some cases and lossy technique better in others.

3. RESEARCH AND DESIGN METHODOLOGY

Improved Dynamic Bit Reduction 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 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 bit reduction decoder to get back the original data. Following are the steps to compress the data with the help of our proposed system.

3.1 COMPRESSION ALGORITHM

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 in ASCII format]

Step VII: Give the output generated by Step VI to Huffman tree to further compress

the data And obtain the result in compressed binary output form.

Step VIII: Display the final result obtained in step VII.

[Output from step VIII is final compressed output]

3.2 DECOMPRESSION ALGORITHM

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.

4. RESULTS AND DISCUSSION

This section describes the results generated by the proposed system. We have taken different data sets (Random, Alphanumeric, Numeral, Special characters) and conducted various experiments to determine the performance of the proposed system.

- Performance Parameters: Performance evaluation of the proposed algorithm is done using two parameters-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.
- 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

5. FINAL PROPOSED SYSTEM FOR RANDOM DATA SET

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

Table Above shows the various experiments conducted by the authors to determine the compression ratio and space saving percentage achieved by the final proposed system for random data set.

5.1 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)	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 5.1, 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.

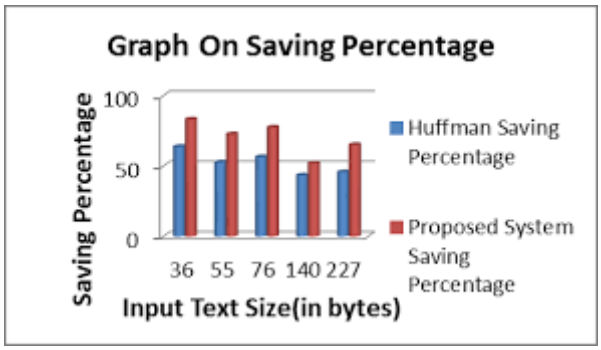

Fig. 5.2 Compression ratio comparison graph for random dataset

The above graph is made on the basis of Table 5.1 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.

6. CONCLUSION AND FUTURE WORK

In this proposed work, an improved dynamic bit reduction algorithm is developed 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. The results obtained by the proposed system are compared with the existing data compression techniques- Bit Reduction and Huffman Coding using parametersCompression Ratio and saving percentage. 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. The existing bit reduction system provides poor compression results. It is based on fixed bit encoding scheme and provides lossy output if special characters are present in the input data. These limitations of the existing Bit reduction system have been overcome by the proposed system as it is using variable Bit encoding scheme. The compression results shown by the proposed system are better than the existing systems (Bit reduction and Huffman coding) as it is using dynamic Bit reduction technique in the first phase and Huffman coding is applied in the second phase to further improve the performance of the proposed system and to achieve better compression results.

FUTURE WORK

Improved Dynamic Bit Reduction 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. In other words, the system works only on ASCII dataset which can be extended to work with Unicode data for future work.