Volume 6, Issue 2, February 2016 ISSN: 2277 128X

International Journal of Advanced Research in Computer Science and Software Engineering

Research Paper

Available online at: www.ijarcsse.com

A Comparative Study of Lossless Data Compression Techniques on Text Data

Achinta Roy
BTech Student, CSE Department
Assam Down Town University, Assam, India

Dr. Lakshmi Prasad Saikia
Professor & Head, CSE Department
Assam Down Town University, Assam, India

Abstract— Data compression is a process that reduces the data size, removing the excessive information and redundancy. Data compression is a common requirement for most of the computerized applications. There are number of data compression algorithms, which are dedicated to compress different data formats.. Hence choosing the best machine learning algorithm is really important. In addition to different compression technologies and methodologies, selection of a good data compression tool is most important. There is a complete range of different data compression techniques available both online and offline working such that it becomes really difficult to choose which technique serves the best. Here comes the necessity of choosing the right method for text compression purposes and hence an algorithm that can reveal the best tool among the given ones. A data compression algorithm is to be developed which consumes less time, while provides more compression ratio as compared to existing techniques. In this paper, we will discuss only the lossless compression techniques and not the lossy techniques as related to our work and the reviews of different basic lossless data compression methods are considered. The methods such as Huffman coding and Arithmetic coding are considered. A conclusion is derived on the basis of these methods.

Keywords— Data compression, Lossless data compression technique, Huffman Coding, Arithmetic coding etc.

I. INTRODUCTION

Data compression is a way to reduce storage cost by eliminating redundancies that happen in most files. There are two types of compression, lossy and lossless. Lossy compression reduced file size by eliminating some unneeded data that won't be recognize by human after decoding, this often used by video and audio compression. Lossless compression on the other hand, manipulates each bit of data inside file to minimize the size without losing any data after decoding. This is important because if file lost even a single bit after decoding, that mean the file is corrupted.

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.

II. LITERATURE SURVEY

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

In 2013, R. S. Brar and B. Singh described 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.

In 2013, S. Porwal et. al, explained on 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.

In 2011, S. Shanmugasundaram and R. Lourdusamy, 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.

In 201, Md. Rubaiyat Hasan introduced a method and system for transmitting a digital image (i.e., an array of pixels) from a digital data source to a digital data receiver. More the size of the data be smaller, it provides better transmission speed and saves time. In this communication we always want to transmit data efficiently and noise free .

In 2013, S. Kaur and V.S. Verma implemented LZW data compression algorithm by finite state machine, thus the text data can be effectively compressed .

In 2015, U. Khuranaand and A. Koul, introduced a new compression technique that uses referencing through two-byte numbers (indices) for the purpose of encoding has been presented. The technique is efficient in providing high compression ratios and faster search through the text.

III. LOSSLESS COMPRESSION TECHNIQUES

A. 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 . 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.

B. Arithmetic Coding

In this method, a code word is not used to represent a symbol of the text. Instead it uses a fraction to represent the entire source message. The occurrence probabilities and the cumulative probabilities of a set of symbols in the source message are taken into account. The cumulative probability range is used in both compression and decompression processes. In the encoding process, the cumulative probabilities are calculated and the range is created in the beginning. While reading the source character by character, the corresponding range of the character within the cumulative probability range is selected. Then the selected range is divided into sub parts according to the probabilities of the alphabet. Then the next character is read and the corresponding sub range is selected. In this way, characters are read repeatedly until the end of the message is encountered. Finally a number should be taken from the final sub range as the output of the encoding process. This will be a fraction in that sub range. Therefore, the entire source message can be represented using a fraction. To decode the encoded message, the number of characters of the source message and the probability/frequency distribution are needed.

IV. MEASUREMENT TO EVALUATE THE PERFORMANCE OF COMPRESSION TECHNIQUES

Depending on the nature of the application there are various criteria to measure the performance of a compression algorithm. When measuring the performance the main concern would be the space efficiency. The time efficiency is another factor. Since the compression behaviour depends on the redundancy of symbols in the source file, it is difficulty to measure performance of a compression algorithm in general. The performance depends on the type and the structure of the input source. Additionally the compression behaviour depends on the category of the compression algorithm: lossy or lossless. If a lossy compression algorithm is used to compress a particular source file, the space efficiency and time efficiency would be higher than that of the lossless compression algorithm. Thus measuring a general performance is difficult and there should be different measurements to evaluate the performances of those compression families. Following are some measurements used to evaluate the performances of lossless algorithms.

A. **Compression Ratio** is the ratio between the size of the compressed file and the size of the source file.

$$compression\ Ratio = \frac{size\ before\ compression}{size\ after\ compression}$$

B. **Compression Factor** is the inverse of the compression ratio. That is the ratio between the size of the source file and the size of the compressed file.

$$compression\ Factor = \frac{size\ after\ compression}{size\ before\ compression}$$

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

$$saving\ percentage = \frac{size\ before\ compression - size\ after\ compression}{size\ before\ compression}\%$$

All the above methods evaluate the effectiveness of compression algorithms using file sizes. There are some other methods to evaluate the performance of compression algorithms. Compression time, computational complexity and probability distribution are also used to measure the effectiveness.

Compression Time

Time taken for the compression and decompression should be considered separately. Some applications like transferring compressed video data, the decompression time is more important, while some other applications both compression and decompression time are equally important. If the compression and decompression times of an algorithm are less or in an acceptable level it implies that the algorithm is acceptable with respective to the time factor.

Evaluating the performance

The performance measurements discussed in the previous section are based on file sizes, time and statistical models. Since they are based on different approaches, all of them can not be applied for all the selected algorithms. Additionally, the quality difference between the original and decompressed file is not considered as a performance factor as the selected algorithms are lossless. The performances of the algorithms depend on the size of the source file and the organization of symbols in the source file. Therefore, a set of files including different types of texts and different file sizes are used as source files. A graph is drawn in order to identify the relationship between the file sizes, the compression time and compression ratio.

Comparing the Performance

The performances of the selected algorithms vary according to the measurements, while one algorithm gives a higher saving percentage it may need higher processing time. Therefore, all these factors are considered for comparison in order to identify the best solution. An algorithm which gives an acceptable saving percentage within a reasonable time period is considered as the best algorithm.

Results

In order to compare the performances of the selected algorithms the compression and decompression times, and compressed file sizes are compared. Figure shows the compression and decompression times, compression ratio and saving percentage of selected files for two the algorithms.

Table I: Comparison of compression and decompression time of Huffman and Arithmetic coding algorithm

FILE NAME	**SIZE (bytes)**	**HUFFMAN CODING**		**ARITHMETIC CODING**	
		COMPRESSION TIME (mili sec)	**DECOMPRESSION TIME (mili sec)**	**COMPRESSION TIME (mili sec)**	**DECOMPRESSION TIME (mili sec)**
Text1.txt	11,252	3766	6750	4465	7178
Text2.txt	15,370	5906	9703	6503	9976
Text3.txt	22,094	16141	16574	17632	17804
Text4.txt	44,355	54719	20606	5919	21457

Table II: Comparison of compression ratio and saving percentage of Huffman and Arithmetic coding algorithm

FILE NAME	**SIZE (bytes)**	**HUFFMAN CODING**			**ARITHMETIC CODING**		
		COMPRESSED FILE	**COMPRESSION RATIO**	**SAVING PERCENTAGE (%)**	**SIZE OF COMPRESSED FILE**	**COMPRESSION RATIO**	**SAVING PERCENTAGE (%)**
Text1.txt	11,252	7,584	1.48	32.60	6764	1.66	39.70
Text2.txt	15,370	8,961	1.59	37.42	7745	1.8	44.43
Text3.txt	22,094	13,826	1.62	38.32	11765	1.805	44.51
Text4.txt	44,355	27,357	1.71	41.70	24568	1.98	49.49

The experimental results of the implemented algorithms, Huffman and arithmetic coding for compression ratio and execution time are depicted in Table I and Table II. As this table shows, on one hand, the compression ratio of the arithmetic coding for different Text sizes is higher than the Huffman coding. On the other hand, arithmetic coding needs more execution time than Huffman coding. This means that the high compression ratio of the arithmetic algorithm is not free. It needs more resources than Huffman algorithm.

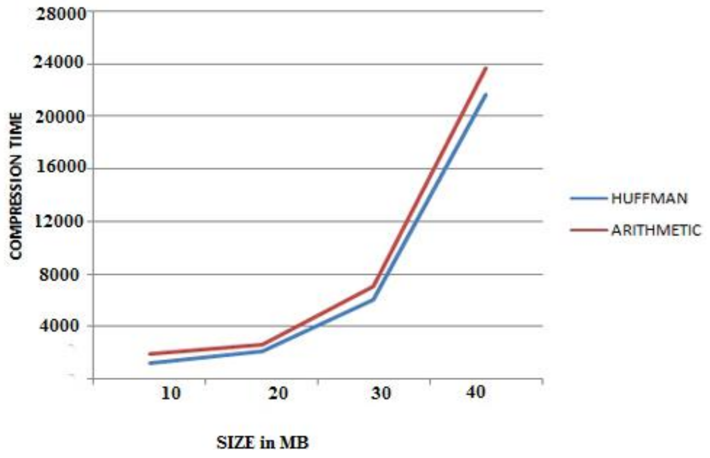

Fig. 1 Comparison of compression time for Huffman and Arithmetic coding algorithms using different text sizes

In Figure 1, by increasing text sizes, the improvement of the compression Time of the Arithmetic coding increases more than the Huffman coding.

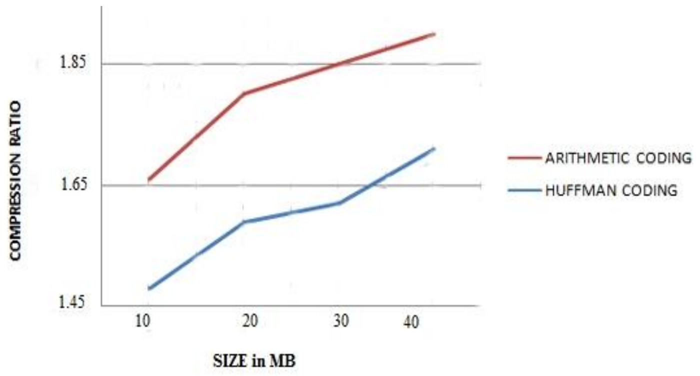

Fig. 2 Comparison of compression ratio for Huffman and Arithmetic coding algorithms using different text sizes

In Figure 2, the compression ratio of Arithmetic coding is greater than Huffman coding.
As shown in this comparison, we can say that the compression ratio of the arithmetic coding for different Text sizes is higher than the Huffman coding. On the other hand, arithmetic coding needs more execution time than Huffman coding.
After the comparison of Huffman and Arithmetic coding, we get some major difference of these two algorithm shown below in table

Table III: Major differences between Arithmetic and Huffman coding.

COMPRESSION METHOD	ARITHMETIC CODING	HUFFMAN CODING
COMPRESSION RATIO	VERY GOOD	POOR
COMPRESSION SPEED	SLOW	FAST
DECOMPRESSION	SLOW	FAST
MEMORY SPACE	VERY LOW	LOW

V. CONCLUSIONS

In this paper, we have find out that arithmetic coding methodology is very powerful over Huffman coding methodology. In comparison we came to know that compression ratio of arithmetic coding is better. But Arithmetic coding takes more compression and decompression time over Huffman coding. If we do not consider time then we can directly say that the Arithmetic coding is better than Huffman coding.

REFERENCES

[1] Introduction to Data Compression, Khalid Sayood, Ed Fox (Editor), March 2000.
[2] Blelloch, E., 2002. Introduction to Data Compression, Computer Science Department, Carnegie Mellon Universit

[3] Pu, I.M., 2006, Fundamental Data Compression, Elsevier, Britain.

[4] 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.

[5] S. Porwal, Y. Chaudhary, J. Joshi and 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.

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

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

[8] Md. R. Hasan*, "Data Compression using Huffman based LZW Encoding Technique"* International Journal of Scientific & Engineering Research, Vol 2, Issue 11, Nov-2011.

[9] S. Kaur and V.S.Verma," Design and Implementation of LZW Data Compression Algorithm", International Journal of Information Sciences and Techniques (IJIST) Vol.2, No.4, July 2012.

[10] 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.

[11] K. Rastogi, K. Segar, "*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.

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