Performance Measurement and Comparison of Lossless Compression Algorithms

Sudhansh Sharma
Jaipuria Institute of Management Studies(JIMS)
Vasundhara,Ghaziabad,U.P.India
sudhansh74@rediffmail.com

Neetu Sharma
Gurukul - The School,
Ghaziabad,U.P. India
neetu.gurukul@gmail.com

Durgansh Sharma
Jaipuria Institute of Management,
Noida, U.P., India
durgansh@gmail.com

ABSTRACT

Data compression is widely required in the era of Information-communication-Technology (ICT), where it can be used to conserve the energy of networks, because a file with reduced size requires less time to get passed over the network. Thus the technique of compression and decompression can be quite effective in establishing efficient communication over the computer networks. The work performed in the paper, compares the Loss less data compression algorithms and analyses various parameters like compression ratio, compression speed, decompression speed, saving percentage. An experimental comparison of a number of different lossless data compression algorithms is presented in this paper. The article is concluded by stating which algorithm performs well for text data.

KEYWORDS

Data compression; Lossy Compression; Lossless Compression; Compression parameters

PREAMBLE

Data compression enters into the field of Information Theory because of its concern with redundancy. Redundant data or information consumes both more space and time, because redundant information in a message takes extra bit to encode, and if we can get rid of that extra information, we will have reduced the size of the message and hence the processing speed. There are various compression techniques to get rid of this redundant information. This paper examines the performance of various compression techniques viz. the Run Length Encoding Algorithm, Huffman Encoding Algorithm, Shannon Fano Algorithm, Adaptive Huffman Encoding Algorithm, Arithmetic Encoding Algorithm and Lempel Zev Welch (LZW) Algorithm. In particular, performance of these algorithms in compressing text data is evaluated and compared.

REVIEW OF LITERATURE

The Compression techniques can be lossless or lossy. Lossy data compression concedes a certain loss of accuracy in exchange for greatly increased compression. Lossy compression proves effective when applied to graphics images and digitized voice. Whereas Lossless compression consists of those techniques guaranteed to generate an exact duplicate of the input data stream after a compress/expand cycle. This is the type of compression used when storing database records, spreadsheets, or word processing files. In these applications, the loss of even a single bit could be catastrophic. Lossless compression techniques like run-length coding [i], Huffman encoding [ii][viii], arithmetic coding [iii], Limpel-Ziv-Welch (LZW) coding [iv] etc. are widely used in compressing medical and satellite images as they retain all information from the original image. Lossy compression techniques like Discrete Fourier Transform (DFT) [v], Discrete Cosine Transform (DCT) [vi], Discrete Wavelet Transform (DWT) [vii] transform the image data to a different domain and quantize the coefficients. These techniques give higher compression ratios.

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 [9]. 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. The Run Length Encoding algorithm uses those runs to compress the original source file while keeping all the non-runs without using for the compression process [xi].

Huffman Encoding

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 [9]. 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[xi].

The Shannon Fano Algorithm

This is another variant of Static Huffman Coding algorithm. The only difference is in the creation of the code word. All the other processes are equivalent to the above mentioned Huffman Encoding Algorithm[xi].

Arithmetic Encoding

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 [x]. 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 [xi].

Lempel Zev Welch Algorithm

Dictionary based compression algorithms are based on a dictionary instead of a statistical model [x]. 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-Zev 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 [xi][xii].

RESEARCH OBJECTIVES

- **To compare and contrast various compression algorithms for different compression performance evaluation parameters**
- **The finding of this paper could create a greater awareness on the choice of the compression algorithm which works best for textual compression.**

RESEARCH METHODOLOGY

In this paper we studied compression ratio, compression time, saving percentage as the parameters to evaluate the effectiveness of compression algorithms using file sizes. Some more parameters to evaluate the performance of compression algorithms are: Compression speed, computational complexity and probability distribution, which are also used to measure the effectiveness.

The performed work involves implementation of various compression algorithms. Further, the text files of various size are processed through the implemented code of the different compression algorithms, and parameters like compressed file size, compression time, decompression time etc are recorded to evaluate various parameters like compression ratio, compression speed, saving percentage etc.

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

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

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.

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

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

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

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.

The performance measurements factors discussed above are based on file sizes, time and statistical models. Since they are based on different approaches, all of them cannot 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 such as English phrases, source codes, user manuals, etc, 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 and decompression time.

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.

ANALYSIS AND INTERPRETATION

Five lossless compression algorithms are tested for ten text files with different file sizes and different contents. Followings are the results for 10 different text files.

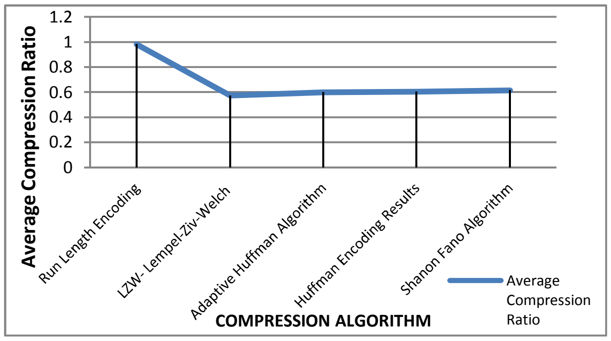
Compression Ratio

After applying compression algorithms on the ten text files, following results are observed for the compression ratio. Table-1 represents the average compression ratio observed for various algorithms. Based on the data recorded in Table-1, below, it is analyzed that LZW algorithm gives excellent compression ratio, where as the RLE algorithm provides the worst of the same. The Graphical representation of the average compression ratio variation is given in Figure -1 below. Where as Figure-2 depicts the observed compression ratio for the entire set of 10 text files under study. Further, Table-2 describes the trend analysis of the compression ratio pattern followed by the various algorithms under the study. Among all algorithms, its observed that compression ratio of the RLE increases with the increase in the original file size.

TABLE – 1: COMPRESSION RATIO – COMPARISON

COMPRESSION ALGORITHM	AVERAGE COMPRESSION RATIO
Run Length Encoding	0.98235853
LZW- Lempel-Ziv-Welch	0.571981384
Adaptive Huffman	0.597829479
Huffman Encoding	0.603261853
Shanon Fano	0.61491948

FIGURE – 1: COMPRESSION RATIO – COMPARISON



FIGURE–2: FILE SIZE Vs COMPRESSION RATIO COMPARISON

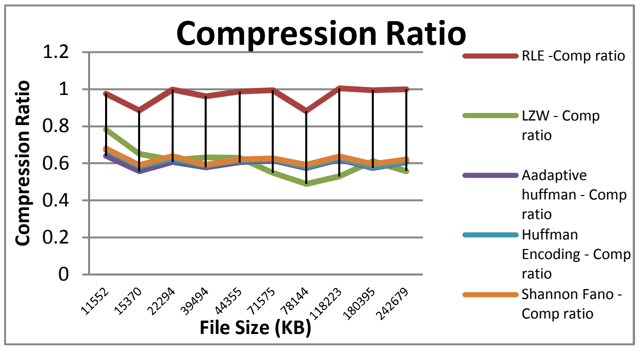


TABLE – 2 : COMPRESSION RATIO – TREND EQUATION & R² VALUE COMPARISON FOR COMPRESSION RATIO

Compression Time

Based on the values tabulated in Table-3, it is identified that RLE consumes least amount of time where Adaptive Huffman algorithm for file compression requires maximum amount of time. This doesn't mean that RLE is the best because, comparing the results of Table 3 and Table-1 for RLE, we observe that the compression ratio is quite poor in case of RLE, which means that compression , which is the basic purpose of the algorithm is performed quickly but not effectively. Whereas the compression ratio of Adaptive Huffman algorithm is observed to be second best among the algorithms under study and this is reflected in its average compression time which is on the higher side. Apart from this LZW's , compression time is slightly lesser than that of Adaptive Huffman and its compression ratio is also slightly better than that of the Adaptive Huffman Algorithm for file compression. Analysing the Trendline equation data and R² Values, given in table-4, we observe that results for LZW and Adaptive Huffman coding are quite close and that of the Huffman coding and Shanon Fano Algorithm are also quite close but results for RLE are not observed to be matching with either of the algorithm under study.

TABLE – 3 : COMPRESSION TIME – COMPARISON

COMPRESSION ALGORITHM	AVERAGE COMPRESSION TIME(SECONDS)
Run Length Encoding	9.3498
LZW- Lempel-Ziv-Welch	314.241
Adaptive Huffman	338.4653
Huffman Encoding	145.0981
Shanon Fano	131.1463

FIGURE –3: COMPRESSION TIME – COMPARISON

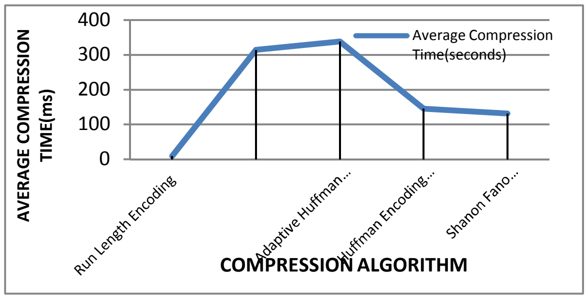
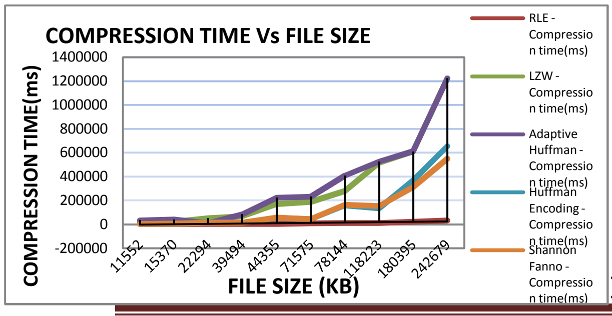


FIGURE – 4:FILE SIZE Vs COMPRESSION TIME



COMPRESSION ALGORITHM	TREND EQUATION TO APPROXIMATE COMPRESSION RATIO	R² VALUE FOR EQUATION TO APPROXIMATE COMPRESSION RATIO
Run Length Encoding	y = 0.004x + 0.942	0.094
LZW- Lempel-Ziv-Welch	y = -0.019x + 0.713	0.536
Adaptive Huffman	y = -0.001x + 0.603	0.009
Huffman Encoding	y = -0.003x + 0.632	0.160
Shanon Fano	y = -0.003x + 0.636	0.105

TABLE – 4: COMPRESSION TIME – TREND EQUATION & R² VALUE COMPARISONVFOR COMPRESSION TIME

COMPRESSION ALGORITHM	TREND EQUATION TO APPROXIMATE COMPRESSION TIME	R² VALUE FOR EQUATION TO APPROXIMATE COMPRESSION TIME
Run Length Encoding	y = 2990x - 7098	0.735
LZW- Lempel-Ziv-Welch	y = 10899x - 28523	0.754
Adaptive Huffman	y = 11086x - 27129	0.792
Huffman Encoding	y = 57060x - 16873	0.666
Shanon Fano	y = 49577x - 14152	0.719

Saving Percentage

Analysis of the values tabulated in Table -5 and Table -6 Adaptive Huffman offers maximum saving percent, apart from this there is a close contest between LZW, Huffman Encoding and Shanon Fano algorithms, but RLE stands ot of the line and offers least Saving percent. Results of Table-6 shows that RLE follows negative slope of trend line, thus it is interpreted that the saving percent declines with the increase in the original file size.

TABLE – 5: SAVING PERCENTAGE – COMPARISON

COMPRESSION ALGORITHM	AVERAGE SAVING PERCENTAGE (%)
Run Length Encoding	3.177851664
LZW- Lempel-Ziv-Welch	39.51374914
Adaptive Huffman	40.12850652
Huffman Encoding	38.95399071
Shanon Fano	38.03128421

FIGURE –5: SAVING PERCENTAGE – COMPARISON

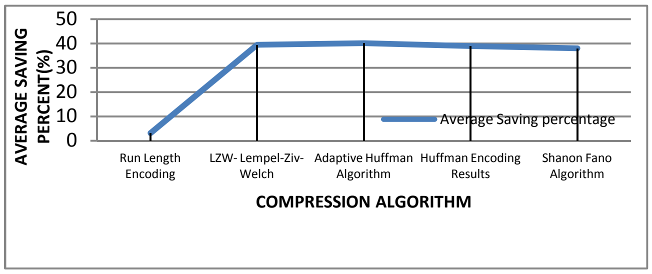
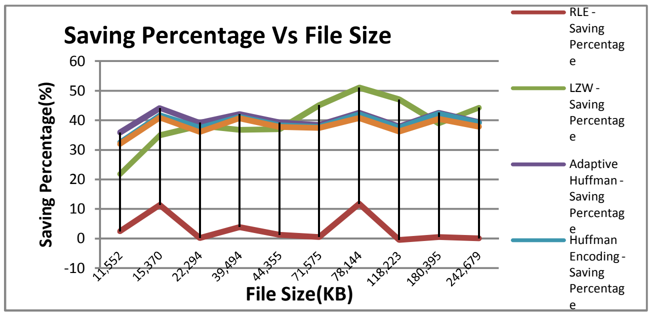


FIGURE – 6:FILE SIZE Vs SAVING PERCENT



VALUE COMPARISON FOR SAVING PERCENT

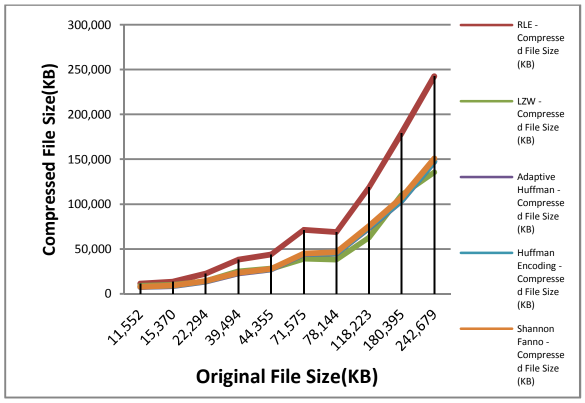
COMPRESSION ALGORITHM	TREND EQUATION TO APPROXIMATE SAVING PERCENTAGE	R² VALUE FOR EQUATION TO APPROXIMATE SAVING PERCENTAGE
Run Length Encoding	$y = -0.467x + 5.750$	0.094
LZW- Lempel-Ziv-Welch	$y = 1.965x + 28.70$	0.536
Adaptive Huffman	$y = 0.081x + 39.68$	0.009
Huffman Encoding	$y = 0.391x + 36.8$	0.160
Shanon Fano	$y = 0.307x + 36.34$	0.105

FINDING AND DISCUSSION

Adaptive Huffman Algorithm needs a relatively larger time period for processing, because the tree should be updated or recreated for both processes. The processing time is relatively small since a common tree for both the processes is used and is created only once. LZW approach works better as the size of the file grows up to a certain amount, because there are more chances to replace identified words by using a small index number.However, it can not be considered as the most efficient algorithm, because it can not be applied for all the cases.

The speed of the Run Length Encoding algorithm is high, but the saving percentage is low for all selected text files. Run Length Encoding algorithm is designed to identify repeating symbols and to replace by a set of characters which indicate the symbol and number of characters in the run. The saving percentage is low for selected text files as there is less number of repeating runs.

FIGURE–7: ORIGINAL FILE SIZE Vs COMPRESSED FILE SIZE FOR DIFFERENT ALGORITHM



Huffman Encoding and Shannon Fano algorithm show similar performances except in the compression times. Huffman Encoding algorithm needs more compression time than Shannon Fano algorithm, but the differences of the decompression times and saving percentages are extremely low. The code efficiency of Shannon Fano
Algorithm is a quite a low value compared to the Huffman encoding algorithm. So the generated code words using Shannon Fano algorithm have to be improved more than the code words of the Huffman Encoding. According to the differences of the compression time Shannon Fano algorithm is faster than the Huffman Encoding algorithm. So this factor can be used to determine the more efficient algorithm from these two.

CONCLUSION

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 for compression and decompression is considered as the best algorithm. Based on the results tabulated in table-7 below, the saving percent parameter recognizes Adaptive Huffman as the best algorithm. But, on grounds of compression & decompression time, plus reasonably acceptable compression ratio makes Shanon Fano as the most suitable algorithm for file compression.

TABLE – 7 : CUMMULATIVE COMPARISON

Compression algorithm	*Compressed File Size-KB*	*Average Comprsession Ratio*	*Compression Time-ms*	*Decompression Time-ms*	*Saving percentage-%*
Run Length Encoding	*80954.3*	*0.98*	*9.35*	*10.54*	*3.18*
LZW	*47107.3*	*0.57*	*314.24*	*391.64*	*39.51*
Adaptive Huffman	*49236.1*	*0.60*	*348.47*	*438.45*	*40.13*
Huffman Encoding	*49683.5*	*0.60*	*145.10*	*200.25*	*38.95*
Shanon Fano	*50643.6*	*0.61*	*131.15*	*146.50*	*38.03*

FUTURE WORK

The Performed work can be extended to evaluate the performance of Lossfull compression algorithms, which are quite useful in image and video compression. Different Lossfull compression algorithms could be implemented and their performance parameters can be evaluated. This comparison could help to choose the most appropriate algorithm among the implemented ones.

REFERENCES

i. Capon J., "A Probabilistic Model for Run-Length Coding of Pictures", IRE Transactions on Information Theory, pp. 157-163, 1959.

ii. Huffman D. A., "A Method for the Construction of Minimum Redundancy Codes", Proceedings of IRE, Vol. 40, No. 10, pp. 1098-1101, 1952.

iii. Abramson N., "Information Theory and Coding", McGraw-Hill, New York, 1963.

iv. Welch T. A., "A Technique for High-Performance Data Compression", IEEE Computer, pp. 8-19, 1984.

v. Freeman A. (translator), Fourier J., "The Analytical Theory of Heat", Cambridge University Press, 1878.

vi. Jayant N. S. and Noll P., "Digital Coding of Waveforms", Prentice Hall, 1984.

vii. Antionini M., Barlaud M., Mathieu P. and Daubechies I., "Image Coding using Wavelet Transform", IEEE Trans. on Image Proc., Vol. 1, No. 2, pp. 205-220, April 1992.

viii. Mark Nelson ,Jean LoupGailly, "The Data Compression", Second Edition,Publisher: IDG Books Worldwide, Inc.,ISBN: 1558514341

ix. Blelloch, E., 2002. Introduction to Data Compression, Computer Science Department, Carnegie Mellon University.

x. Campos, A.S.E, Basic arithmetic coding by Arturo Campos Website, Available from:http://www.arturocampos.com/ac_arithmetic.html.

xi. S.R. Kodituwakku , U. S.Amarasinghe," Comparison Of Lossless Data Compression Algorithms for Text Data ", Vol 1 No.4 pp 416-425,2008, Indian Journal of Computer Science and Engineering , ISSN : 0976-5166

xii. R. Rajeswari, R. Rajesh," WBMP Compression"International Journal of Wisdom Based Computing, Vol. 1(1), 2011,pp 50-53