Empirical and Statistical Evaluation of the Effectiveness of Four Lossless Data Compression Algorithms

N. A. Azeez*, A. A. Lasisi

Department of Computer Sciences, University of Lagos, Nigeria.

ABSTRACT: Data compression is the process of reducing the size of a file to effectively reduce storage space and communication cost. The evolvement in technology and digital age has led to an unparalleled usage of digital files in this current decade. The usage of data has resulted to an increase in the amount of data being transmitted via various channels of data communication which has prompted the need to look into the current lossless data compression algorithms to check for their level of effectiveness so as to maximally reduce the bandwidth requirement in communication and transfer of data. Four lossless data compression algorithm: Lempel-Ziv Welch algorithm, Shannon-Fano algorithm, Adaptive Huffman algorithm and Run-Length encoding have been selected for implementation. The choice of these algorithms was based on their similarities, particularly in application areas. Their level of efficiency and effectiveness were evaluated using some set of predefined performance evaluation metrics namely compression ratio, compression factor, compression time, saving percentage, entropy and code efficiency. The algorithms implementation was done in the NetBeans Integrated Development Environment using Java as the programming language. Through the statistical analysis performed using Boxplot and ANOVA and comparison made on the four algorithms, Lempel Ziv Welch algorithm was the most efficient and effective based on the metrics used for evaluation.

KEYWORDS: Data compression, lossless, evaluation, entropy, algorithm

[Received July 11 2016; Revised December 13 2016; Accepted December 20 2016]

I. INTRODUCTION

The need for data and information sent through various means of communication needs to be depressed to a reduced and yet compact form in very important. Compression of data is the process of reducing the size of a data into a smaller but yet a compact form. It is also the process of sinking large storage of data in a way of reducing its communication cost. Data compression which is also known as source coding revolves around the reduction of bits in the original file size as compared to the original state.

There are two forms of data compression; Lossless data compression which exploits redundancy in a text data to represent the data in a compact form without data loss e.g. text data. Lossy data compression allows for the loss of data during the process of compression.

In 1970s, software compression came to live in the advent of Internet and subsequently online storage with the Huffman encoding (invented by David Huffman who was studying information theory at MIT) which is similar to Shannon-Fano coding but different as its probability tree is built in a top-down form(Mohammed and Ibrahiem , 2007). Abraham Lempel and Jacob Ziv in 1977 came up with Lempel-Ziv algorithm which was the first algorithm to use dictionary in compressing data (Arup, et al., 2013). Since then, many variants of Lempel-Ziv algorithm have grown from LZ77, LZ78, LZMA and LZX for which most have faded after its invention.

The advent of this various compression techniques begs for the need to evaluate Lempel-Ziv Welch algorithm, Shannon-Fano algorithm, Adaptive Huffman algorithm and Run-Length encoding for a proper test on their efficiency and effectiveness.

Against this backdrop, this work aims at providing comprehensive details on the effectiveness and efficiency of the algorithms base on the selected metrics for their evaluation.

II. DESCRIPTION OF LOSSLESS COMPRESSION ALGORITHMS

A. Entropy Based Encoding

This type of lossless data compression algorithm tallies the number of occurrence of each character/symbol in the original document. These unique characters are represented with a new set of symbol generated by the algorithm. The length of the newly generated symbols depends on the level of occurrence of each symbol in the original document (Kodituwakkuand Amarasinghe, 2015). Entropy based encoding algorithm is also based on the statistical information of the source file – looking at the rate of occurrence of a particular character (Manas, et al., 2012). An example of this algorithm is Shannon Fano encoding.

Entropy is the randomness of occurrence for a set of string at a particular time.

Entropy can be defined as:

$$H(s) = \sum(P(S)log_2 \frac{1}{P(S)}) \quad (1)$$

*Corresponding author's e-mail address: nazeez@unilag.edu.ng

doi: http://dx.doi.org/10.4314/njtd.v13i2.4

(Wang, 2011) where "S" is the set of probable states, and P(S) is the likelihood of state

$$P(S) = \frac{\text{Compressed File Size}}{\text{Length of Chracter}} \quad (2)$$

B. Adaptive Huffman Encoding

Huffman encoding algorithm was invented by David Huffman in the year 1951. This algorithm is an entropy based algorithm mainly for lossless data compression. Character of fixed length codes are substituted with variable length codes. Huffman Encoding Algorithm is the process of using the probability of occurrence of a symbol in the original source document to create a code word for each character(Tamanna and Sonia , 2014). Adaptive Huffman algorithm which is a branch of Huffman Encoding algorithm creates a tree in a bottom up form during the process of calculating characters occurrence (Pooja, et al., 2015).

C. Shannon Fano Coding

Shannon Fano data compression algorithm was named after Claude Shannon and Robert Fano after their efforts to create an encoding procedure that will generate a binary code treein a top-down form (Kannanand Murugan, 2012). The algorithm which is entropy based and similar to Huffman encoding algorithm evaluates a characters reoccurrence and allocates a code word with corresponding code length.

D. Dictionary Based Encoding

This algorithm is also known as substituting encoding. It holds a data structure called "dictionary" which contains strings. The encoder of the algorithm in the process of compression matches a substring in the original file to the string in the dictionary (Manas, et al., 2012). If a match is found, the encoder replaces the substring with a reference to the dictionary.

E. Lempel Ziv Welch

Lampel Zev Welch was named after Abraham Lampel and Jacob Zev worked on an LZ78 algorithm in 1977; Terry Welch modified it in 1984 for implementation in an extraordinary performance disk (Pooja, et al., 2015). It is a substitution compression algorithm which creates an active dictionary with a set of strings and thereby substitutes each corresponding substring in the original files with the string in the dictionary. The string in the dictionary acts as a reference to the substring in the original document.

F. Run Length Encoding

Run Length encoding can be regarded as the simplest lossless data compression algorithm. It processes a document on number of "Runs" and "Non-Runs" (Shrusti, et al., 2013). It simply counts the number of times a character occurs repeatedly in the source file, for example, BOOKKEPPER will be encoded as 1B2O2K1E2P1E1R. (Sebastian, 2003).

III. RELATED WORKS

Arup, et al. (2013) presented a paper which was set with the objective of examining the performance of various lossless data compression based on different test files. Various metrics were used to determine the level of performance of each algorithm. Three lossless data compression algorithm, namely Huffman encoding, Shannon Fano and LampelZiv Welch (LZW) were implemented and examined. From the various performance evaluation metrics carried out (compression ratio, compression factor, entropy and code efficiency), LZW was said to be slower, Shannon Fano has a higher average decompression time. It was concluded that depending on the various performance metrics, their performance varies. It was recommended that more Lossy and lossless data compression algorithm be examined in future while they should also be tested on larger test files.

Barath, et al. (2013) designed software, Domain "Sun Zip" developed with Java programming language with the aim of reducing the number of bit and byte representation of a character. The software works by reducing the bit representation of source file, lessens the disk storage space of such data and thereby allows easy transmission over a network. It was noted that other third party software such as WinRAR, WinZip etc. poses some disadvantages and difficulties. The software was developed using a lossless data compression algorithm named Huffman encoding Algorithm. Some major drawbacks were identified in the previous existing third party software which are; Data insecurity, higher compression time and monopoly in file extension.

It was observed by SubhamastanRao, et al. (2011) that speed (processing time) is the main challenge during the separate process of data compression and encryption. The paper focused on the need to combine these two processes together thereby lessening the challenges. The idea behind this combination was to add to data compression a pseudo random shuffle. Shuffling of nodes in the tree of Huffman algorithm is done to produce a single mapping of the Huffman table. Decompression cannot be done once the Huffman table is encrypted thus simultaneous encryption and compression is achieved.

Challenges facing the separate process of compression and encryption ranges from low sped, acquiring more cost and the computer having more processing time. These challenges were the main reason behind combining compression and encryption algorithms. Execution time of both process reduced drastically and the new algorithm was deemed as good as other common algorithm such as DES, RC5, etc.

The approach improved the speed and also provided more security. Enhancement is encouraged on this approach to achieve more efficiency and the algorithm was said to be prone to security attack. Hanaa, et al. (2015) observed that images contain multiple redundancies from high correlation between pixels which occupies a lot of space. Many algorithms have been designed and developed to compress images. This

research was based on analyzing all the image compression algorithms and identifying the advantages and shortfalls. The main objective of this research was to find a way of reducing the amount of power consumed by redundant images.

In the source data, three major types of data redundancy were observed; Spatial redundancy, temporal redundancy and spectral redundancy. Various processes involved in its image compression included mapper, quantizer and entropy encoding. The performance metrics used to measure the level of efficiency of image compression were quality of image, compression ratio, power consumption and speed of compression which can be divided into two; computational complexity and memory resources. During the course of evaluation, it was reached that SPIHT is the best technique due to its compactness and generation of low bit rate. Adaptation of SPIHT for Wireless Media Sensor Network (WMSN) was encouraged as an area to be researched upon.

Suarjaya (2012) proposed a new data compression algorithm "J BIT ENCODING" (JBE) which manipulates every bit in a source file to minimize the data size without losing any information. The algorithm was considered to be a lossless data compression algorithm. The developed algorithm was also compared with other algorithms to measure the level of effectiveness and efficiency.

Other algorithms used for the comparison are Run Length encoding, Burrows wheeler transform, Move to Front (MTF) and Arithmetic coding. The proposed algorithm with other four algorithms were tested with five different data files. The results were inconclusive due to the hybrid nature of test files used e.g. document content included audio, text, and video. The author recognizes the need for more review and research into J Bit encoding algorithm.

Lempel ZivWelch which was "incorporated as the Standard of the consultative committee on International telegraphy and telephony" was implemented with a little modification. Simrandeep and Sulochana in 2012 designed the dictionary of the algorithm based on "content addressable memory array". Xilinx ISE simulation tool was used to derive accurate performance measures. The algorithm which was evaluated by a finite state machine technique achieved a compression rate of 30.3% with 60.25% reduction in disk storage. The result of the developed LempelZiv Welch data compression algorithm assigned 5 bit to each character instead of 7 bits. Various test data were used for the analysis.

Pooja, et al. (2015) proposed a two stage data compression algorithm OLZWH which used both Lempel Ziv Welch and Adaptive Huffman encoding algorithm at the optimal level. In the algorithm, dictionaries are formed for input character symbols in two modes; set of indices and set of ASCII. OLZW was applied to set of indices while Adaptive Huffman was applied to ASCII code. The analysis were however unclear as there is no detailed explanation and statistical interpretation of the results obtained.

IV. DATA COMPESSION EVALUATION TECHNIQUES/METRICS.

Various performance evaluations metric were used to evaluate the four lossless data compression algorithms. The implication of these values with respect to -114 dBm defined by FCC as the criteria of the empty spaces for TV white space (Nasir et. al., 2013) is that FCC has chosen additional sensing margins of 27.3 dB and 3.3 dB in both cases of channel 31, but the margin is 2.7 dB in the case of channel 10.

A. Compression Ratio

This was calculated by finding the ratio between the compressed and original file.

$$\text{Compression Ratio} = \frac{Compressed\ File\ Size}{Source\ Document\ Size} \quad (3)$$

Source: Kodituwakkuand Amarasinghe, 2015

B. Compression Factor

This is the inverse of compression ratio which can be calculated as:

$$\text{Compression Factor} = \frac{Source\ document\ size}{Compressed\ file\ Size} \quad (4)$$

C. Saving Percentage

According to Kodituwakku and Amarasinghe (2015),

$$\text{Saving Percentage} = \frac{Source\ Document\ size - Compressed\ File\ Size}{Source\ Document\ Size}\% \quad (5)$$

D. Compression and Decompression Time

This calculates the time taken for each algorithm to compress file of a particular size and also to decompress same file back to its original form. The time will be calculated in Nanoseconds (Ns).

E. Entropy

Generally, entropy refers to disorder or uncertainty. Entropy is used if the data compression algorithm is based on statistical information of the source file. Two events happen in a source document; an event that occurs rarely and the other which occurs repeatedly. Entropy can be calculated (Kodituwakku and Amarasinghe, 2015) as:

$$H(s) = \sum(P(S) log_2 \frac{1}{P(S)}) \quad (6)$$

where S is the set of probable states, and P(S) is the likelihood of state.

F. Code Efficiency

Code efficiency can be defined as the percentage in ratio between the source file entropy and the average code length of the source file. It can be calculated as:

$$E = \frac{H(S)}{L} \quad (7)$$

where E is the code efficiency, H(S) is the entropy and L is the average code length.

Source: Kodituwakku and Amarasinghe, 2015.

G. Average Code Length

This can be defined as the average number of bits expected to represent a single code word. For the length of the code word in the source file is known, the average code length can be calculated as (Kodituwakkuand Amarasinghe, 2015):

$$L = \sum p, l \quad (8)$$

where p is the likelihood of occurrence of a particular symbol; l is the length of a code word for a particular symbol.

V. IMPLEMENTATION, FINDINGS AND RESULTS

Four lossless data compression algorithms; two of Entropy based data compression algorithm (Adaptive Huffman compression algorithm and Shannon Fano compression algorithm), Run Length encoding data compression algorithm based on repetitive and redundancy values and a Dictionary based data compression algorithm - Lempel Ziv Welch have been implemented with Java programming language in the NetBeans Integrated Development Environment (IDE) and are tested against 10 text data with varied sizes. The data files are in 145813 bytes, 3814642 bytes, 96166 bytes, 147456 bytes, 242819 bytes, 27031 bytes, 62976 bytes, 451793 bytes, 200438 bytes and 2928078 bytes. The test data also varies in content as some are programming languages codes, numbers, eBooks, previous past project and normally text data. The text files are with the extension .doc, .docx, .txt, .pdf and .rtf

Also graphics and audio documents are tested for Adaptive Huffman and Lempel Ziv Welch data compression algorithm. The files are of 63101 bytes, 4568712 bytes, and 1122430 bytes for the graphics in .jpeg, .gif and .jpg format and 8340775 bytes, 2279529 bytes for audio in .mp3 extension. Shannon Fano and Run Length data compression algorithms do not work well for graphics and audio files. This has been done to determine the algorithm with the most maximal level of efficiency and effectiveness. Table 1 provides analysis of the four lossless compression algorithms using various metrics for performance evaluation.

Going by the result in Table 1, Run Length compression algorithm did not work well with the test data. Run Length works well on repeated character and since all the data have little or few repeated values, the compressed data increased from that of the original data which isn't the desired result expected. The compressions ratio and factor are over the mark while the saving percentage is negative all through. In File 1, the compressed file size almost doubled the original file size.
LampelZiv Welch data compression algorithm makes use of a dynamic dictionary. The result in Table 2 shows a very good compression ratio. File 10 of Table 2 gives a saving percentage of 78.19%. All the files compressed have a reduction in size as compared to Run Length which increased in size. The lowest saving percentage is 26.59%.The compression ratio and factor of all files are quite good. The saving percentage is still positive in the compression of picture and graphics. The compression time is also within satisfaction. With this algorithm, communication cost and storage space will be reduced.

Implementation of Adaptive Huffman algorithm as shown in Table 3 shows a dynamic tree for the traversal of nodes with a relatively average saving percentage. The saving percentage for the text document was as high as 63.53%. The algorithm doesn't work well with tabs as the compression of .docx file has shown a low saving percentage. For example, File 3 has 0.21% while File 8 has- 0.13%. Adaptive Huffman compression ratio of picture and audio file is very high as shown in File 11 to File 15. The saving percentage for audio is a bit higher than that of picture. Adaptive Huffman helps in reducing file size of compressed data which helps to reduce communication cost and storage space.

Shannon Fano which is a variant of Huffman Algorithm has quite been known not to have a better code efficiency to Adaptive Huffman. Results obtained as shown is Table 4 gives all the files compression ratio to be above 100% which isn't efficient. The saving percentage is also in the negative state. The compression factor is far low while the entropy is in the

Table 1: Results for Run length encoding algorithm base of the metrics used

Original/Source File				Compressed File				
File No	File Type	File Size (byte)	No of Characters	File Size (byte)	Compression Ratio	Compression Factor	Saving Percentage	Compression Time (Ns)
1	Text.txt	145813	119498	280096	192.092612	0.520582	-92.092612	117805320
2	Text.pdf	3814642	190632	3960402	103.821066	0.963	-3.821066	28574764776
3	Text.docx	96166	134812	194907	202.677662	0.493394	-102.677662	265030951
4	Text.doc	147456	119903	245432	166.444227	0.600802	-66.444227	46255635
5	Text.pdf	242819	28735	451610	185.986270	0.537674	-85.986270	61262819
6	Text.txt	27031	21743	52942	195.856609	0.510578	-95.856609	74430256
7	Text.doc	62976	27542	87153	138.390816	0.722591	-38.390816	43526813
8	Text.docx	451793	885240	578932	128.140985	0.780390	-28.140985	76893549
9	Text.rtf	200438	40530	381690	190.427962	0.525133	-90.427962	77116678
10	Text.rtf	2928078	1985544	5747092	196.275224	0.509489	-96.275224	404530274

Table 2: Results for Lempel Ziv Welch algorithm base on the metrics used metrics used.

Original/Source File				Compressed File				
File No	**File Type**	**File Size (byte)**	**No of Characters**	**File Size (byte)**	**Compression Ratio**	**Compression Factor**	**Saving Percentage**	**Compression Time (Ns)**
1	Text.txt	145813	119498	58537	40.145255	2.490954	59.85	1818280253
2	Text.pdf	3814642	190632	2160402	56.634463	1.765709	43.37	18574764776
3	Text.docx	96166	134812	66322	68.966163	1.449986	31.03	901233950
4	Text.doc	147456	119903	47058	31.913249	3.133495	68.08	1612183652
5	Text.pdf	242819	28735	78166	32.191056	3.106453	67.80	1517335876
6	Text.txt	27031	21743	12257	45.344234	2.205352	54.66	307013484
7	Text.doc	62976	27542	25080	39.824695	2.511005	60.18	414828744
8	Text.docx	451793	885240	289979	64.184040	1.558019	35.82	2309066572
9	Text.rtf	200438	40530	66263	33.059100	3.024886	66.94	818822983
10	Text.rtf	2928078	1985544	638574	21.808640	4.585339	78.19	12503148589
11	Picture .jpg	63101	NA	46323	73.410881	1.362196	26.59	1336731046
12	Picture.jpeg	4568712	NA	2834401	62.039389	1.611879	37.96	27124451524
13	Picture.gif	1122430	NA	683645	60.907584	1.641832	39.09	4373213534
14	audio.mp3	8340775	NA	4627141	55.476152	1.802576	44.52	40555952684
15	audio.mp3	2279529	NA	1386837	60.838752	1.643689	39.16	11450106158

Table 3: Results for Adaptive Huffman Algorithm base on the metrics used.

Original/Source File				Compressed File				
File No	**File Type**	**File Size (byte)**	**No of Characters**	**File Size (byte)**	**Compression Ratio**	**Compression Factor**	**Saving Percentage**	**Compression Time (Ns)**
1	Text.txt	145813	119498	90001	61.723577	1.620126	38.28	297175620
2	Doc.pdf	3814642	190632	3647056	36.47056	1.045951	63.53	1081429949
3	Text.docx	96166	134812	95971	99.797225	1.002032	0.21	51938673
4	Text.doc	147456	119903	79477	53.898790	1.855329	46.11	331428722
5	Text.pdf	242819	28735	155586	64.074887	1.560674	35.93	81607140
6	Text.txt	27031	21743	18014	66.642003	1.500555	33.36	32092003
7	Text.doc	62976	27542	44807	71.149327	1.405495	28.86	30690993
8	Text.docx	451793	885240	452383	100.1306	0.998696	-0.13	146173646
9	Text.rtf	200438	40530	138969	69.332661	1.442322	30.67	129791786
10	Text.rtf	2928078	1985544	1859653	63.511047	1.574529	36.49	714729168
11	Picture .jpg	63101	NA	63530	100.679862	0.993247	-0.67	190259256
12	Picture.jpeg	4568712	NA	4565140	99.921816	1.000782	0.08	1381189336
13	Picture.gif	1122430	NA	1123425	100.088647	0.999114	-0.08	436710648
14	audio.mp3	8340775	NA	8171960	97.976027	1.020658	3.03	2243053531
15	audio.mp3	2279529	NA	2274598	99.783683	1.002168	0.22	716609976

Table 4: Results for Shannon Fano algorithm base on the metrics used.

Original/Source File				Compressed File						
File No	**File Type**	**File Size (byte)**	**No of Characters**	**File Size (byte)**	**Compressi on Ratio**	**Compressi on Factor**	**Saving Percentage**	**Compression Time (Ns)**	**Entropy**	**Code Efficiency %**
1	Text.txt	145813	119498	213557	146.459506	0.682783	-46.459506	632008452	6.2761	82.8753
2	Text.pdf	3814642	190632	5160402	135.278802	0.739214	-35.278802	37574764776	6.7655	84.6129
3	Text.docx	96166	134812	2578961	268.178046	0.037289	-168.178046	6660258056	7.9999	75.9012
4	Text.doc	147456	119903	158347	107.385932	0.931220	-7.385932	685743578	7.3829	84.8726
5	Text.pdf	242819	28735	356273	146.723691	0.681553	-46.723691	728021577	7.0912	80.8452
6	Text.txt	27031	21743	49894	184.580667	0.541769	-84.580667	189032759	7.7432	82.0126
7	Text.doc	62976	27542	117696	186.890244	0.535073	-86.890244	779840189	7.8921	82.8721
8	Text.docx	451793	885240	764562	169.228386	0.590917	-69.228386	867324476	7.8921	83.8710
9	Text.rtf	200438	40530	354252	176.738942	0.565806	-76.738942	314077893	7.9232	80.1939
10	Text.rtf	2928078	1985544	4744516	162.035164	0.617150	-62.035164	4104947185	7.6729	82.8907

VI. COMPARISON OF THE FOUR LOSSLESS DATA COMPRESSION ALGORITHMS

The four lossless data compression algorithms which results have been shown in Table 2 were compared based on their saving percentage, compression ratio, compression time, entropy and code efficiency. With the comparison shown in Table 5 and graphical comparison result in Figure 1, it is shown that Lempel Ziv Welch clearly has a better saving percentage than the other algorithm compared though Adaptive Huffman has a better saving percentage in Text2.pdf only.

Table 5: Comparison of the four lossless algorithms based on saving.

File Type	RUN LENGTH	LEMPEL ZIV WELCH	ADAPTIVE HUFFMAN	SHANNON FANO
Text1.txt	-92.092612	59.85	38.28	-46.459506
Text2.pdf	-3.821066	43.37	63.53	-35.278802
Text3.docx	-102.677662	31.03	0.21	-168.178046
Text4.doc	-66.444227	68.08	46.11	-7.385932
Text5.pdf	-85.98627	67.8	35.93	-46.723691
Text6.txt	-95.856609	54.66	33.36	-84.580667
Text7.doc	-38.390816	60.18	28.86	-86.890244
Text8.docx	-28.140985	35.82	-0.13	-69.228386
Text9.rtf	-90.427962	66.94	30.67	-76.738942
Text10.rtf	-96.275224	78.19	36.49	-62.035164
Picture11 .jpg	NA	26.59	-0.67	NA
Picture12.jpeg	NA	37.96	0.08	NA
Picture13.gif	NA	39.09	-0.08	NA
audio14.mp3	NA	44.52	3.03	NA
audio15.mp3	NA	39.16	0.22	NA

The closer the compression ratio is to "1%", the more efficient the algorithm is. In the result shown in Table 6 and its graphical representation in Figure 2, Lempel Ziv Welch algorithm has a better compression ratio in all test data except in Text2.pdf where Adaptive Huffman algorithm has a better compression ratio. It can be deduced that Lempel Ziv Welch has a better compression ratio to other algorithm.

Table 7 shows the Analysis of Variance which was used to deduce that there are significant difference in the mean value of each of the algorithms. The above boxplot graph shows Lempel Ziv Welch algorithm with a better saving percentage.

A. Comparison Based on Compression Ratio

The closer the compression ratio is to "1%", the more efficient the algorithm is. In the result shown in Table 6 and its graphical representation in Figure 2, Lempel Ziv Welch algorithm has a better compression ratio in all test data except in Text2.pdf where Adaptive Huffman algorithm has a better compression ratio. It can be deduced that Lempel Ziv Welch has a better compression ratio to other algorithm.

B. Comparison Based On Compression Time

In the result shown in Table 7 and its graphical representation in Figure 3, Adaptive Huffman has a better compression time. The average compression rate of 524,325,363.1 Nanoseconds is regarded as the best. Lempel Ziv Welch algorithm which has a better compression ratio and saving percentage has the least good average compression time of 8,374,475,588 Nanoseconds.

C. Comparison between Original and Compressed File Sizes

In the result comparison showed at Table 8 and its graphical representation in Figure 4, the original file sizes are compared with the their corresponding compressed file sizes. Lempel Ziv Welch algorithm has the lower rate of compressed file size as compared to other in all test files.

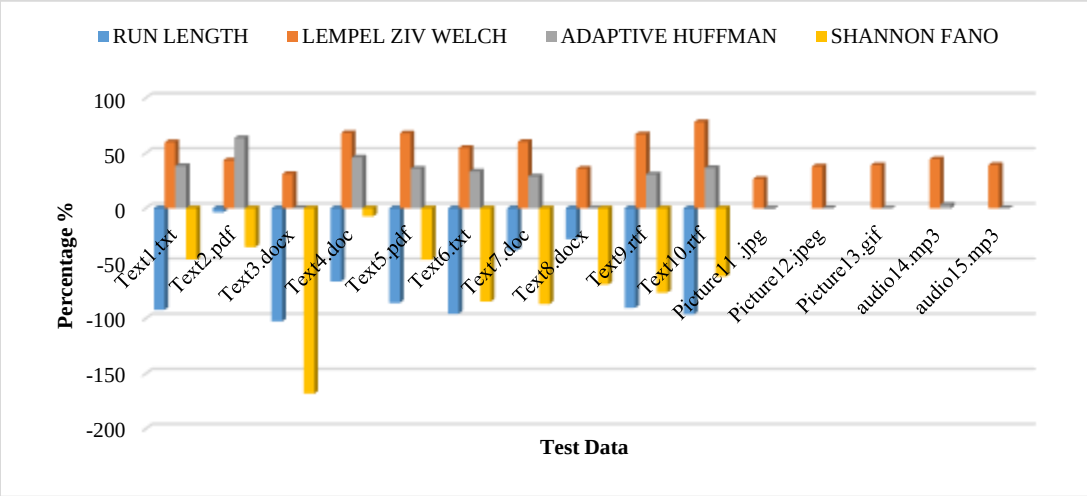

Figure 1: Graphical Comparison of saving percentage for the four lossless algorithms.

Table 6: Comparison of the four lossless algorithms based on saving

File Type	RUN LENGTH	LEMPEL ZIV WELCH	ADAPTIVE HUFFMAN	SHANNON FANO
Text1.txt	117805320	1818280253	297175620	632008452
Text2.pdf	28574764776	18574764776	1081429949	37574764776
Text3.docx	265030951	901233950	51938673	6660258056
Text4.doc	46255635	1612183652	331428722	685743578
Text5.pdf	61262819	1517335876	81607140	728021577
Text6.txt	74430256	307013484	32092003	189032759
Text7.doc	43526813	414828744	30690993	779840189
Text8.docx	76893549	2309066572	146173646	867324476
Text9.rtf	77116678	818822983	129791786	314077893
Text10.rtf	404530274	12503148589	714729168	4104947185
Picture11 .jpg	NA	1336731046	190259256	NA
Picture12.jpeg	NA	27124451524	1381189336	NA
Picture13.gif	NA	4373213534	436710648	NA
audio14.mp3	NA	40555952684	2243053531	NA
audio15.mp3	NA	11450106158	716609976	NA
AVERAGE	2,974,161,707.Ns	8,374,475,588 Ns	524,325,363.1 Ns	5,253,601,894Ns

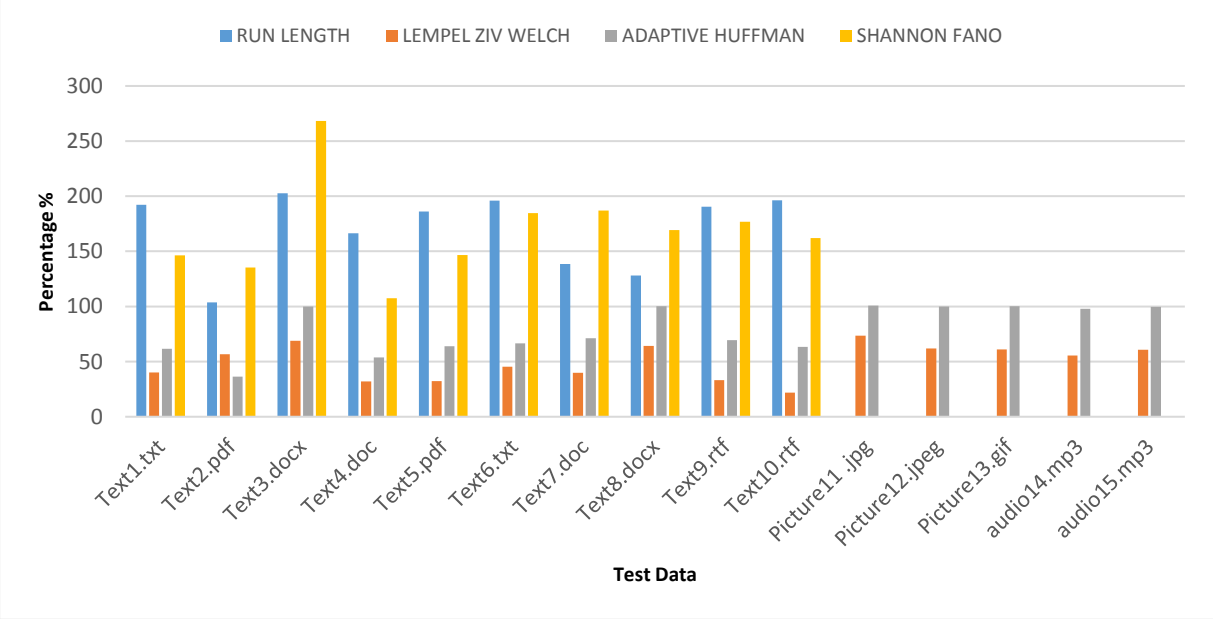

Figure 2: Graphical comparison of compression ratio of the four lossless algorithms.

VI. CONCLUSION

A study and evaluation of four lossless data compression algorithm was done. The algorithms were implemented and tested with different test data of different sizes. A comparison of all four algorithms was done to know their level of efficiency and effectiveness. By working on their result analysis and graphical representation while considering the compression factor, compression ratio, saving percentage and ability to compress audio and graphics file effectively, the Lempel Ziv Welch algorithm which is based on using dictionary is considered to be the most effective and efficient of the four data compression algorithm evaluated. The result and values are very good and acceptable. Since an efficient and effective compression algorithm has been identified this in turn allows optimal usage of storage space and also reduction in communication cost. Great knowledge has been contributed to the world of computer science as an efficient data compression algorithm has been identified.

A system should be put in place that will recognize a file type and subsequently assign it to a suitable data compression algorithm. Research should be focused towards Context Mixing Algorithm such as PAQ which is efficient in its compression ration but slow due to usage of multiple statistical prototypes. The speed should be improved upon. Use of compression via substring enumeration (CSE), a compression technique should be research more into to improve its level of efficiency.

Table 7: Comparison of the four lossless algorithms based on saving

File Type	RUN LENGTH	LEMPEL ZI V WELCH	ADAPTIVE HUFFMAN	SHANNON FANO
Text1.txt	117805320	1818280253	297175620	632008452
Text2.pdf	28574764776	18574764776	1081429949	37574764776
Text3.docx	265030951	901233950	51938673	6660258056
Text4.doc	46255635	1612183652	331428722	685743578
Text5.pdf	61262819	1517335876	81607140	728021577
Text6.txt	74430256	307013484	32092003	189032759
Text7.doc	43526813	414828744	30690993	779840189
Text8.docx	76893549	2309066572	146173646	867324476
Text9.rtf	77116678	818822983	129791786	314077893
Text10.rtf	404530274	12503148589	714729168	4104947185
Picture11 .jpg	NA	1336731046	190259256	NA
Picture12.jpeg	NA	27124451524	1381189336	NA
Picture13.gif	NA	4373213534	436710648	NA
audio14.mp3	NA	40555952684	2243053531	NA
audio15.mp3	NA	11450106158	716609976	NA
AVERAGE	2,974,161,707.Ns	8,374,475,588 Ns	524,325,363.1 Ns	5,253,601,894Ns

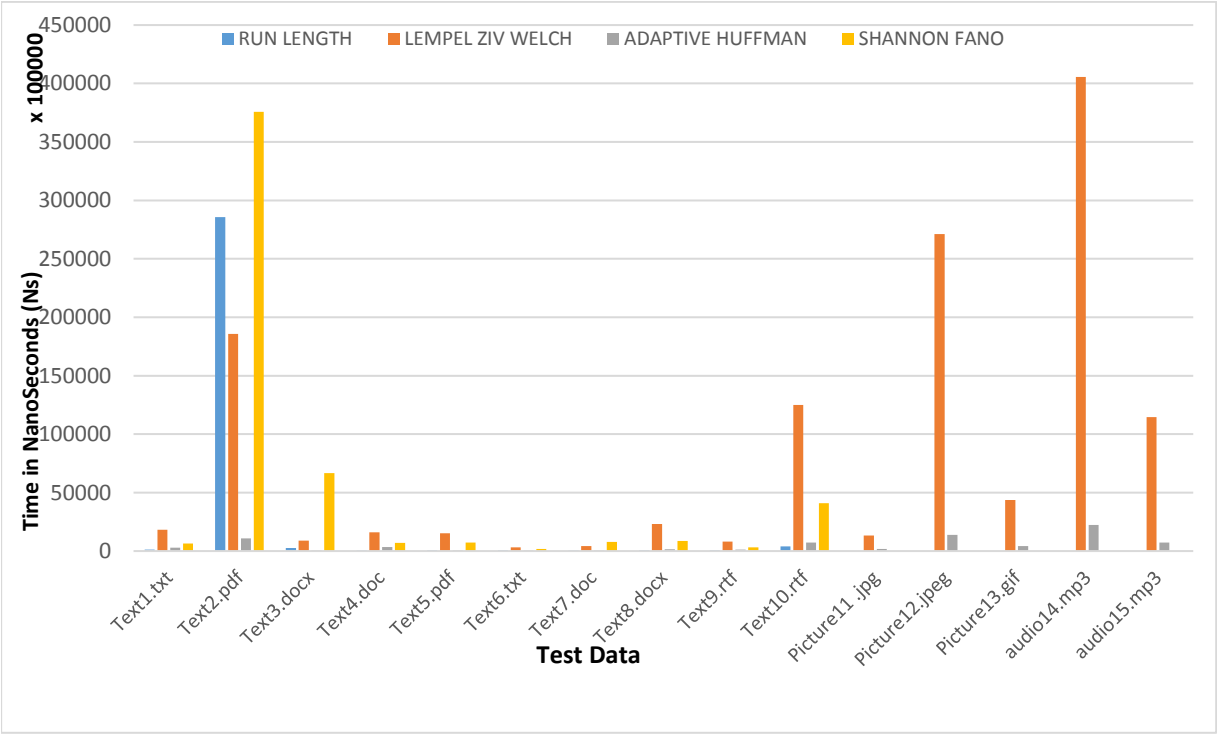

Figure 3: Graphical comparison of compression time of the four lossless algorithms.

Table 8: Comparison between original file size and compressed file size of the four lossless algorithms.

ORIGINAL FILE SIZES (bytes)	RUN LENGTH (COMPRESSED)	LEMPEL ZIV WELCH (COMPRESSED)	ADAPTIVE HUFFMAN (COMPRESSED)	SHANNON FANO (COMPRESSED)
145813	280096	58537	90001	213557
3814642	3960402	2160402	3647056	5160402
96166	194907	66322	95971	2578961
147456	245432	47058	79477	158347
242819	451610	78166	155586	356273
27031	52942	12257	18014	49894
62976	87153	25080	44807	117696
451793	578932	289979	452383	764562
200438	381690	66263	138969	354252
2928078	5747092	638574	1859653	4744516
63101	NA	46323	63530	NA
4568712	NA	2834401	4565140	NA
1122430	NA	683645	1123425	NA
8340775	NA	4627141	8171960	NA
2279529	NA	1386837	2274598	NA

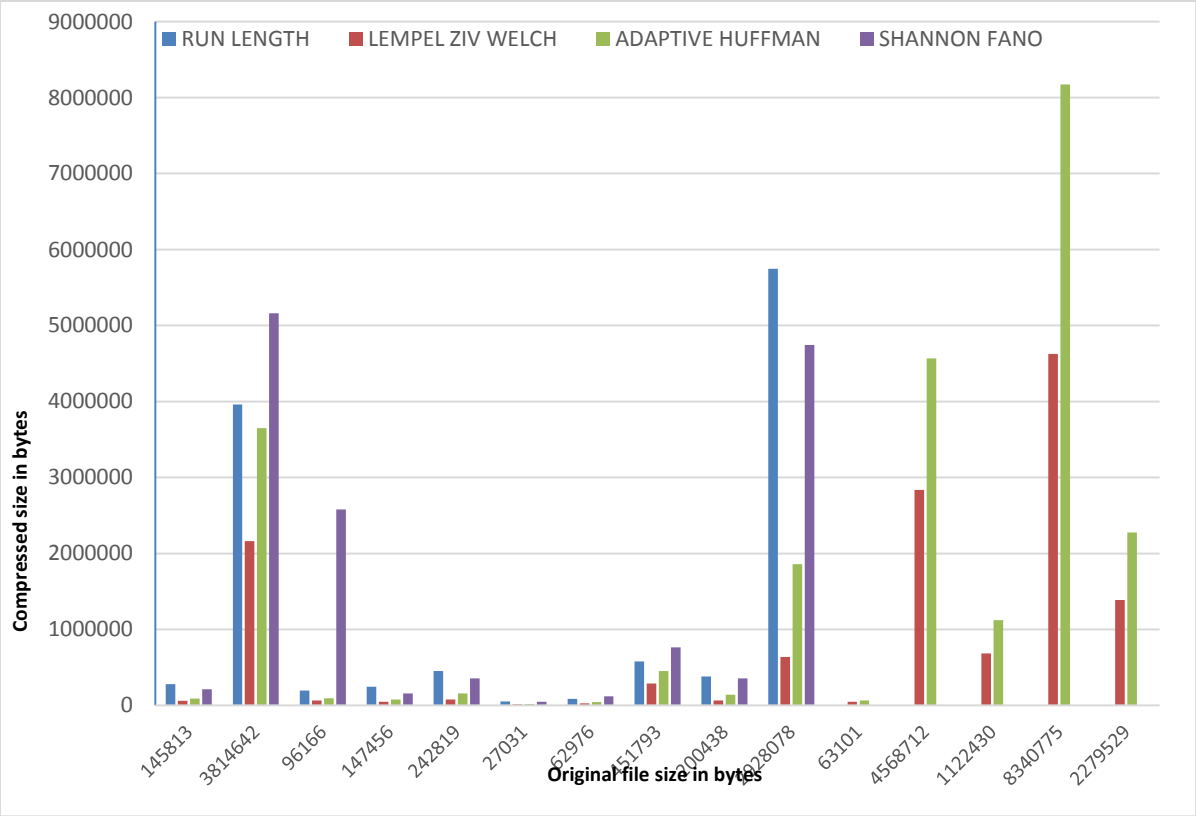

Figure 4: Graphical comparison of original file size against compressed file size.

REFERENCES

Anandhi, G. G. and Satthiyaraj, S., (2013). VC Using Lempel-Ziv-Welch Algorithm. International Journal of Engineering and Advanced Technology (IJEAT), 2(3): 60 - 64.

Anitha, S., (2015). Lossless image compression and decompression using huffman coding. International Research Journal of Engineering and Technology (IRJET), 2(1): 240 - 247.

Arup, K. B.; B. Tanumon and A. Saheb (2013). Comparison Study of Lossless Data Compression Algorithms for Text Data. IOSR Journal of Computer Engineering (IOSR-JCE), 11(6): 15-19.

Barath, C. K.; M. K. Varun and T. Gayathri (2013). Technique of Data Analysis and File Compression Using

Huffman Algorithm. International Journal of Advanced Research in Computer Science and Software Engineering, 3(11): 346 - 348.

Blelloch, G. E., (2013). Introduction to Data Compression. Berkeley: Computer Science Department, Carnegie Mellon University.

Hanaa, Z.; A. E. Mostafa and A. A. Hesham (2015). Image compression algorithms in wireless multimedia sensor networks: A survey. Ain Shams Engineering Journal, 6: 481 - 490.

Howard, P. G., (1993). The Design and Analysis of Efficient Lossless Data Compression Systems. Rhode Island: Department of Computer Science, Brown University.

Kannan, E. and Murugan, G., (2012). Lossless Image Compression Algorithm for Transmitting Over Low Bandwidth Line. International Journal of Advanced Research in Computer Science and Software Engineering, 2(2): 242 - 256.

Kodituwakku, S. and Amarasinghe, U. S., (2015). Comparison of Lossless data compression algorithms for text data. Indian Journal of Computer Science and Engineering, 1(4): 416-425.

Manas, K. M.; K. M. Tapas and K. Alok, K (2012). Parallel Lempel-Ziv-Welch (PLZW) Technique for Data Compression. (IJCSIT) International Journal of Computer Science and Information Technologies, 3(3): 4038 - 4040.

Mohamed, S. S. and Kumar, P. S., (2013). Evaluating Effectiveness of Data Transmission and Compression Technique in Wireless Sensor Networks. International Journal of Advanced Research in Computer Science and Software Engineering, 3(1): 70 - 73.

Mohammed, A. and Ibrahiem, E. M., (2007). Comparative Study between Various Algorithms of Data Compression Techniques.in Proceedings of the World Congress on Engineering and Computer Science, San Francisco, USA. 326-336.

Pooja, J.; J. Anurag and A. Chetan (2015). Improving data compression ratio by the use of optimality of LZW and adaptive Huffman algorithm (OLZWH). International Journal on Information Theory (IJIT), 4(1): 11 - 19.

Pooja, S. (2015). Lossless data compression techniques and comparison between the algorithms. International Research Journal of Engineering and Technology (IRJET), 2(2): 383 - 386.

Sebastian, D. (2003). Universal lossless data compression algorithms. Silesian University of Technology Faculty of Automatic Control, Electronics and Computer Science Institute of Computer Science. Poland: Silesian University of Technology. PhD Thesis, 1-214.

Shrusti, P.; C. Yashi and M. J. Jitendra (2013). Data Compression Methodologies for Lossless Data and Comparison between Algorithms. International Journal of Engineering Science and Innovative Technology (IJESIT), 2(2): 142 - 147.

Simrandeep, K. and Sallekhana, V. V., (2012). HDL implementation of data compression: LZW algorithm. International Journal of Advanced Technology and Engineering Research (IJATER), 2(2): 115 - 120.

Suarjaya, A. D., (2012). A New Algorithm for Data Compression Optimization. (IJACSA) International Journal of Advanced Computer Science and Applications, 3(8): 14 - 17.

SubhamastanRao, T.; M. Soujanya, T. Hemalatha and T. Revathi (2011). Simultaneous Data Compression and Encryption. International Journal of Computer Science and Information Technologies, 2(5): 2369-2374.

Tamanna, G. and Sonia, V., (2014). Research paper on enhancing data compression rate using steganography. International Journal of Computer and Mathematical Sciences, 3(4): 29 - 40.

Wang, W.-Y., (2011). A Unique Perspective on Data Coding and Decoding. Entropy, 13: 53 - 63.