8th IEEE International Conference Humanoid, Nanotechnology, Information Technology
Communication and Control, Environment and Management (HNICEM)
The Institute of Electrical and Electronics Engineers Inc. (IEEE) – Philippine Section
9-12 December 2015 Water Front Hotel, Cebu, Philippines

Comparison of Huffman Algorithm and Lempel-Ziv Algorithm for Audio, Image and Text Compression

Rhen Anjerome Bedruz[#1], Ana Riza F. Quiros[#2]
Gokongwei College of Engineering
De La Salle University
Manila, Philippines
[1]rhen_anjerome_bedruz@dlsu.edu.ph
[2]ana_riza_quiros@dlsu.edu.ph

***Abstract*— In digital communications, it is necessary to compress the data for a faster and more reliable transmission. As such, the data should undergo source encoding, also known as data compression, which is the process by which data are compressed into a fewer number of bits, before transmission. Also, source encoding is essential to limit file sizes for data storage. Two of the most common and most widely used source encoding techniques are the Huffman Algorithm and Lempel-Ziv Algorithm. The main objective of this research is to identify which technique is better in text, image and audio compression applications. The files for each data type were converted into bit streams using an analog-to-digital converter and pulse code modulation. The bit streams underwent compression through both compression algorithms and the efficiency of each algorithm is quantified by measuring their compression ratio for each data type.**

***Index Terms*— data compression, Huffman Algorithm, Lempel-Ziv Coding, text file, image file, audio file**

I. Introduction

Digital communications deals with the transmission and reception of digital data. These types of data can contains the information needed to be transmitted. It can be in different formats, it can be in text, image, video, documents, or audio form. Before transmitting these information, it must be converted into digital format which gives a stream of bits which is either a logic 1 or 0. The bit stream normally comprises of several bits which most of the time reaches for more millions of bits. The rate for a single bit transmission varies on the medium which it is transmitted. For example, an ordinary copper wire can transmit for up to 11.2 Million bits per second. Thus, it will take minutes for the transmission of very large files.

There is always the need to transmit these information completely at a given time. If the data to be transmitted is large, it might not reach the destination at a specific time. In order to compensate with this scenario, source encoding is used. Source encoding is a technique used to reduce the average number of bits used for a given file. This is used so that data will be compressed in order to maximize the data transmission.

There are many types of source coding techniques such as Huffman Algorithm, Lempel-Ziv algorithm, or Shannon-Fano Elias Algorithm. Between these algorithms, the most commonly used is the Huffman and Lempel-Ziv algorithm because of the low average code word length they produce which gives them high compression ratios. The Huffman algorithm is capable of achieving the shortest possible code word length for a particular symbol which depends on the probability while Lempel-Ziv coding is done via symbol-by-symbol encoding depending on the sequence of the symbols encoded [1].

Studies on data compression regularly encounter the problem of the efficiency of the algorithm used. As such, several trials and runs have been made before finding the most appropriate algorithm to use. Hence, effort and time has been used in finding the right algorithm instead of optimizing their methodology. In order to optimize the data transmission, the proponents investigated the source coding algorithm that must be used for each of data format presented.

II. Data Compression Algorithms

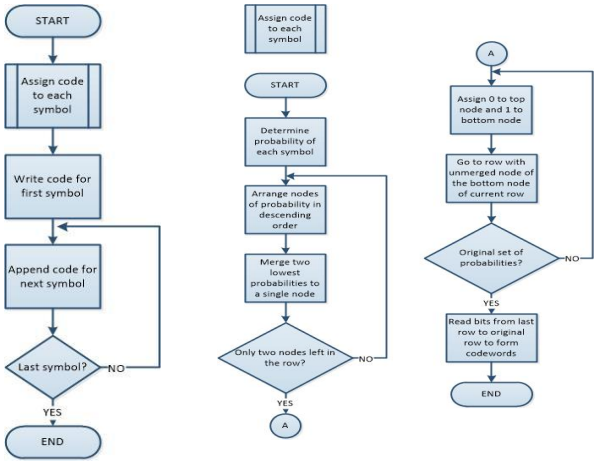
The main goal of digital communications is to transmit information completely at a given time. This goal is hard to achieve if the data being transmitted is large because larger data take more time to be transmitted completely [1]. In order to compensate with this scenario, source encoding was created to promote data compression. Source encoding is a technique used to reduce the average number of the number of data bits per unit time. Through source encoding, the data will be compressed in order to maximize the channel allocated for the data. It is also used to increase the capacity for the data transmission. There are many source coding techniques available, namely Huffman Algorithm, Lempel-Ziv Algorithm, Shannon-Fano Elias Encoding, Arithmetic Coding, and Adaptive Coding [2].

978-1-5090-0360-0/15/$31.00 ©2015 IEEE

The primary focus of this study are the applications of two source encoding techniques on different types of data format such as audio, video, and text: the Huffman and the Lempel-Ziv Algorithms. These algorithms were commonly used due to the low average code word length they produce which gave a high compression ratio.

A. Huffman Algorithm

The Huffman code is a prefix-free, variable-length code which is capable of achieving the shortest possible code word length for a particular symbol which can be greater than its entropy. This algorithm is done by assigning the most occurring symbol by the least number of bits up to the least occurring symbol with the most number of bits [3].



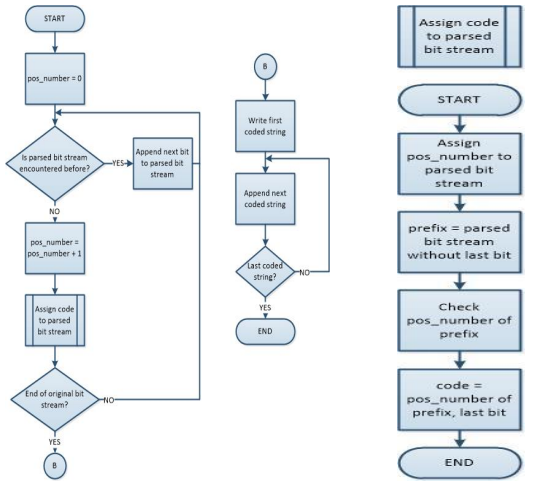
(a) General Process Flow (b) Assignment of Codes

Fig. 1. Huffman Algorithm Data Compression Flowchart

A study aimed to compress GPS data used the Huffman Algorithm [4]. GPS data is in ASCII code which contains information such as latitude, longitude, altitude, speed, date and time. Another study proposed its own compression method and compared it with the Huffman Algorithm using various file types [5]. The Huffman algorithm was also used for image by steganography by concealing an image inside another image through the use of Huffman Algorithm. This was used for security purposes or for transmission of secret data. Using Huffman algorithm, the image was concealed better making it better over other existing algorithms [6]. Huffman algorithm was also used for equalization. Equalization is performed by balancing the contrast, brightness and sharpness of the picture [7]. When Lempel-Ziv algorithm was used, it showed a promising result which can be compared to entropy encoding which is said to be good for images [8]. Moreover, it yielded a better compression ratio when modified using human visual system algorithm [9]. The advantage of using Huffman for audio is that it does not remove high frequency components unlike other algorithms [10]. The same algorithm can be applied in wireless sensor networks which have a variable data stream [11].

B. Lempel-Ziv Algorithm

Lempel-Ziv on the other hand, does not depend on the probability of occurrence of the source symbol. This follows a universal source coding procedure that does not depend on the statistical model of the source. This algorithm is done via symbol-by-symbol encoding depending on the sequence of the symbols encoded. Because of the effectiveness of this method, this was commonly used for hard disk drive compression on personal computers [12].



(a) General Process Flow (b) Assignment of Codes

Fig. 2. Lempel-Ziv Algorithm Data Compression Flowchart

A study on language-based text file compression proposed its own compression method and compared its performance against existing techniques such as the Huffman Algorithm and the Adaptive LZW Method [13]. A variant of the Lempel-Ziv Algorithm called Statistical Lempel-Ziv Compression Algorithm was used in a study to compress text and C files [14]. A study intended to design and fabricate a prototype high-speed VLSI chip to meet the speed and software requirements for data compression implemented Lempel-Ziv Algorithm for text compression [15]. In another study, this algorithm also possesses good random access and one-pass compression which is suitable for text since it does not rely on the probability distribution of the data [16]. When Lempel-Ziv algorithm was used, it showed a promising result which can be compared to entropy encoding which is said to be good for images [17]. Moreover, it yielded a better compression ratio when modified using human visual system algorithm [18]. In the contrast, it is significantly slower which tends to remove the high frequency component the audio possesses [19]. By using Lempel-Ziv algorithm, the researchers proved that lossy data compression schemes are universal for stationary and individual sources [20]. Finally, the Lempel-Ziv algorithm was

used for data on very large scale integration (VLSI). This deals with large data sets and they proved that Lempel-Ziv can deal with this kind of data [21].

A study used both Huffman Algorithm and Lempel-Ziv Coding in the compression of the analog data from the output of a sensor [22].

III. Methodology

A. File Description

This research involved the compression of three types of data namely text, image and audio using two techniques which are the Huffman Algorithm and Lempel-Ziv Coding. Ten files of varying sizes for each data type were compressed. The size of each file for two of the data types (text and audio) were patterned based on the conversion of the byte sizes of the data used in Jalilian, Haghighat and Rezvanian's study namely 1024, 2048, 3072, 4096, 5120, 6144, 7168, 8192, 9216, and 10240 bytes. However, the sizes of the image files were based on the sizes of the most commonly used photos in image processing they are labelled as the standard test images by Gonzales and Woods.

The following were the sources of the files: the text files were from essays; the image files were from built-in pictures in Matlab and open-source websites; and the audio files were from bomb explosion sounds. Past studies involving text and audio compression only included the file names and their descriptions in their write-ups. Because of the inaccessibility of the files used in their researches and unavailability of open-source text and audio files, such files were gathered from the internet and were subjected to editing so as to conform to the file size limitations indicated in Table 1.

TABLE I. Size Limitations of Each Data Type

Data Type	Minimum Size Limit	Maximum Size Limit
Text	1,176 characters	10,198 characters
Image	1 layer	3 layers
Audio	2 seconds	20.5 seconds

To pattern with Jalilian's group's study, excerpts from essays were used for text files because they are normalized texts with randomized probabilities for each character. Bomb explosion sounds, as used by Ezhilarasan's research team in their study, were the audio files used because they are often used as sample audio for data communication studies. The image files used were built-in Matlab pictures because they are used most often for past researches involving image compression.

B. Formatting of Information

Before compression, the data were converted first to digital format. For the conversion of text, each text was converted into ASCII format and the pseudocode is shown on figure 3a. For image data, it was done using histogram function of Matlab. The algorithm for image formatting is shown on figure 3b. For the audio data, it used the PCM encoding whose algorithm is shown on figure 3c. The techniques for information formatting was created by the researchers themselves by combining MATLAB functions.

```
Begin
  n = 1
    input "string"
    for i = 1st char to last char of
    "string"
        Dig = convert i to ASCII
        X = size of digital text
        for j = 1 to X
           stream(1,n) = Dig(1,j)
           n = n + 1
        end
    end
End
```

(a) Pseudocode of the text information formatting

```
Begin
   n = 1
   read image file (img)
   x,y,z is size of (img)
   for 1 to z
     for 1 to y
       for 1 to x
         strm(1,n)= dec to bin
              (img(x,y,z))
         n = n + 1
       end
     end
   end
End
```

(b) Pseudocode of the Image Infromation formatting

```
Begin
   n = 1
   read audio file (aud)
   reso = (max(a)-min(a))/256
   diga = pcm of aud at reso
   (a,b) size of diga
    for 1 to a
       for 1 to b
         strm(1,n)= dec to bin
              (diga(a,b))
         n = n + 1
       end
    end
end
```

(c) Pseudocode of the Audio Information formatting

Fig. 3. Information Formatting Pseudocode

C. Data Compression

Compression took place after the data has been converted to a bit stream by the ADC. The two techniques that were used for compression were the Huffman Algorithm and the Lempel-Ziv Coding. Both algorithms were encoded using the Matlab software using the flowcharts shown in Figures 1 and 2.

D. Measurement of Performance Metric

The performance of the two data compression algorithms were compared through their compression ratio and compression time for each data type. The following criteria will be used in choosing the more efficient data compression algorithm: (1) lower mean compression ratio, (2) smaller standard deviation of compression ratio, (3) lower mean compression time; and (4) smaller standard deviation of compression time.

The written codes counted the number of bits of the original bit sequence and the compressed bit sequence for the computation of the compression ratio. The ratio of the original number of bits after the ADC and the compressed number of bits after each data compression technique determined the compression ratio. The sizes of the original and compressed data are measured in terms of bits.

The compression time was determined by identifying the run time of the program of each compression algorithm.

IV. Data and Results

After the researchers simulated the data compression algorithms, the following results were obtained from the simulation.

A. Text Files

Table 2 shows the dictionary for each text file made by the Huffman Algorithm and the Lempel-Ziv Algorithm. As

observed, as the file size or the number of characters in the text increase, the number of symbols in the Huffman Algorithm's dictionary also increases. However, for larger file sizes, the number of symbols becomes almost constant because for larger texts, almost all possible characters were used. The table also shows the number of symbols in the dictionary made by the Lempel-Ziv Algorithm. As observed from the table, the number of symbols continuously increases as the file size is increased as opposed to that of the Huffman Algorithm which reaches a point where the number of symbols becomes unchanging. That is because the creation of symbols by the Lempel-Ziv is derived from parsing.

TABLE II. HUFFMAN AND LEMPEL-ZIV DICTIONARY FOR TEXT

Text File	Number of Characters	Number of Symbols	
		Huffman	Lempel-Ziv
File I	1176	37	933
File II	2341	49	1807
File III	3092	56	2259
File IV	4096	35	2620
File V	5120	35	3142
File VI	6144	35	3647
File VII	7168	35	4152
File VIII	8192	35	4641
File IX	9198	63	5525
File X	10198	63	6018

Table 3 summarizes the performance of each compression algorithm by calculating the compression ratio using Equation 1 and determining the compression time. As shown, the Huffman Algorithm had consistently compressed the data. However, for the smaller text files, the Lempel-Ziv Algorithm expanded the data instead of compressing it. As the file size increased, it performed compression on the data. However, its compression ratios were still much higher compared to that of the Huffman Algorithm. For most text files, the Huffman Algorithm compressed texts faster.

The table compares the performance metric of both compression algorithms for text files namely the compression ratio and compression time using the mean and standard deviation. As seen, the Huffman Algorithm has a lower mean and standard deviation for both the compression ratio and compression time.

TABLE III. PERFORMANCE OF HUFFMAN AND LEMPEL-ZIV

Text File	Original Number of Bits	Huffman			Lempel-Ziv		
		Compressed Number of Bits	Compression Ratio (%)	Compression Time	Compressed Number of Bits	Compression Ratio (%)	Compression Time
I	8232	5004	60.79	0.92	9240	112.24	0.80
II	18728	10167	54.29	1.22	19637	104.85	1.42
III	24736	13398	54.17	1.11	25272	102.16	1.74
IV	28672	17839	62.22	0.89	29965	104.50	1.15
V	35840	22300	62.22	1.05	36751	102.54	1.38
VI	43008	26236	61.00	1.21	43485	101.10	1.66
VII	50176	31247	62.27	1.55	49937	99.52	1.98
VIII	57344	35715	62.28	1.72	56783	99.02	2.22
IX	73584	40152	54.56	1.72	69159	93.98	2.53
X	81584	44439	54.47	2.03	76061	93.23	2.86
		Mean	58.8273	1.3464		101.3188	1.7777
		Standard Deviation	3.6729	0.3667		5.2154	0.6012

B. Image Files

Table 4 shows the dictionary of the Huffman Algorithm and the Lempel-Ziv Algorithm for each image file. As observed, the number of symbols in Huffman Algorithm's dictionary remains at 256 at varying images. The table also displays the dictionary generated by the Lempel-Ziv Algorithm for the image files. As the image file increase in the number of layers, the number of symbols generated through parsing increases as well.

TABLE IV. HUFFMAN AND LEMPEL-ZIV DICTIONARY FOR IMAGE

Text File	Number of Layers	Number of Symbols	
		Huffman	Lempel-Ziv
File I	1	256	240
File II	1	256	211
File III	1	256	241
File IV	2	256	429
File V	2	256	570
File VI	2	256	387
File VII	2	256	500
File VIII	2	256	428
File IX	3	256	830
File X	3	256	911

Table 5 summarizes the performance of the two compression algorithms using their compression ratio and compression time for each image file. By observing the table, it can be said that the compression ratios of the Huffman Algorithm are consistently higher than that of the Lempel-Ziv Algorithm for all ten image files. However, the difference between their compression ratios is fluctuates as the image file

layers increases. For the compression time, the Huffman Algorithm has a consistent very high compression time compared to the Lempel-Ziv algorithm. The table compares the performance metric of both compression algorithms for image files namely the compression ratio and compression time using the mean and standard deviation. As seen, the Huffman Algorithm has a higher mean and standard deviation for both the compression ratio and compression time.

TABLE V. PERFORMANCE OF HUFFMAN AND LEMPEL-ZIV

Image File	Original Number of Bits	Huffman			Lempel-Ziv		
		Compressed Number of Bits	Compression Ratio (%)	Compression Time	Compressed Number of Bits	Compression Ratio (%)	Compression Time
I	2097152	1478084	70.48	12.80	975360	46.50	3.40
II	2097152	1491700	71.12	12.80	961974	45.87	3.43
III	2097152	1434784	68.41	13.23	979968	46.72	3.40
IV	4194304	2032449	48.45	26.42	1289059	30.73	10.75
V	4194304	2561497	61.07	25.89	1789810	42.67	10.28
VI	4194304	3159369	75.32	25.62	1527723	36.42	10.23
VII	4194304	2863006	68.25	26.17	1749998	41.72	10.23
VIII	4194304	3218026	76.72	25.86	1285648	30.65	10.50
IX	6291456	4559119	72.46	38.94	2072019	32.93	15.48
X	6291456	4126153	65.58	39.25	2452901	38.98	15.75
		Mean	67.79	24.70		39.32	9.34
		Standard Deviation	7.75	9.15		6.03	4.35

C. Audio Files

Table 6 shows the dictionary of the Huffman Algorithm and the Lempel-Ziv Algorithm for each audio file. As observed, the number of symbols in Huffman Algorithm's dictionary increases as the audio becomes longer. However, this value saturates to about 16 at larger audio files because there are only 16 symbols represented by the 16 quantized levels by the ADC. The table also displays the dictionary generated by the Lempel-Ziv Algorithm for the audio files. As the audio file becomes longer, the number of symbols generated through parsing increases as well.

TABLE VI. HUFFMAN AND LEMPEL-ZIV DICTIONARY FOR AUDIO

Audio File	Audio Length (seconds)	Number of Symbols	
		Huffman	Lempel-Ziv
File I	2	5	342
File II	4.1	7	903
File III	6.1	8	1333
File IV	8.2	9	1758
File V	10.2	13	1773
File VI	12.3	15	2336
File VII	14.3	16	2396
File VIII	16.4	16	2940
File IX	18.4	15	3347
File X	20.5	15	3807

Table 7 summarizes the performance of the two compression algorithms for each audio file. Comparing directly the means and standard deviation of the compression ratio and the compression time is insufficient to formulate a conclusion. Upon performing a one-tailed t-test on the unpaired compression ratios, the t-statistic at 99% level of confidence (2.602) is less than the t-calculated (2.883). It can be concluded that the mean of the compression ratios of Huffman is significantly lower than Lempel-Ziv's. However, performing a two-tailed t-test for the compression time showed that the t-statistic at 95% confidence (1.740) is greater than the t-calculated (0.7007) which results to the acceptance of the null hypothesis that there is no significant difference between the means of the compression times of the two algorithms.

TABLE VII. PERFORMANCE OF HUFFMAN AND LEMPEL-ZIV

Audio File	Original Number of Bits	Huffman			Lempel-Ziv		
		Compressed Number of Bits	Compression Ratio (%)	Compression Time	Compressed Number of Bits	Compression Ratio (%)	Compression Time
I	8000	2162	27.03	1.35	2909	36.36	1.19
II	16400	5314	32.40	1.23	8910	54.32	1.70
III	24400	8251	33.82	1.31	13949	57.16	1.86
IV	32800	11384	34.71	1.83	19049	58.07	2.02
V	40800	13727	33.64	2.01	19229	47.12	2.67
VI	49200	20099	40.85	2.29	26273	53.40	2.67
VII	57200	24484	42.80	2.49	27053	47.29	2.54
VIII	65600	32295	49.23	2.69	34125	52.01	2.90
IX	73600	39242	53.32	3.63	39416	53.55	4.07
X	82000	47345	57.74	4.07	45357	55.31	4.37
		Mean	40.56	2.29		51.47	2.61
		Standard Deviation	9.57	0.92		6.10	0.95

V. CONCLUSION

For text files, due to lower compression ratio and lower compression time, the Huffman Algorithm is recommended. For text files, Lempel-Ziv showed very little to no compression at all. For the image files, Lempel-Ziv is better compared to Huffman Algorithm because of the huge difference between their compression ratios and compression times. The reason is that when dealing with images, it uses a very large data which

takes time for the computation of the probabilities unlike in Lempel-Ziv. For audio files, comparison between the compression time of the two algorithms was inconclusive because no significant difference between the two. However, on the basis of their compression ratio, the Huffman Algorithm compressed audio files more efficiently than the Lempel-Ziv Algorithm. The differences between the performance of the Huffman Algorithm and Lempel-Ziv Algorithm among different data types could have been from the process by which each algorithm creates their dictionary which also affects the number of symbols created. Also, the formatting of the data and the data's inherent characteristics can also give rise to the number of symbols created.

VI. Recommendation

Future research on data compression may opt to use a more diverse set of files to be used. More sets of files per data type and larger file sizes can be used. Other file types - documents, videos, and C files to name a few - can be the subject of future studies. A PCM encoder which uses a larger number of bits for encoding can be used for the ADC to not limit the quantized levels to 16 only. Moreover, variations of the Huffman Algorithm and Lempel-Ziv Algorithm such as the Adaptive Huffman Algorithm and Lempel-Ziv-Welch Algorithm can be investigated by future researchers as well.

References

[1] B. Sklar, Digital Communications: Fundamentals and Applications, NJ: Prentice Hall, 2001.

[2] H. Nguyen and E. Shwedyk, Introduction to Digital and Data Communicaions, New York: Campbridge, 2009.

[3] U. Madhow, Fundamentals of Digital Communications, New York: Campbridge, 2008.

[4] R. Gonzales and R. Woods, Digital Image Processing, New Jersey: Prentice Hall, 2001.

[5] S. Kwong and Y. Ho, "A statistical Lempel-Ziv compression algorithm for personal digital assistant (PDA)," IEEE Transactions on Consumer Electronics, vol. 47, pp. 154-162, Feb. 2001.

[6] M. Ezhilarasan, P. Thambidurai, K. Praveena, and S. Srinivasan, "A new entropy encoding technique for multimedia data compression," in International Conference on Computational Intelligence and Multimedia Applications, 2007, pp. 157-161.

[7] J. Adiego, G. Navarro and P. D. L. Fuente, "Lempel-Ziv compression of structured text," in Data Compression Conference, Spain, 2004.

[8] R. Das and T. Tuithung, "A novel steganography method for image based on Huffman Encoding," in Emerging Trends and Applications in Computer Science (NCETACS),3rd National Conference , Shillong, 2012.

[9] L. Flores-Pulido, M. C. L. Olivares-Gonzalez, M. Osorio and O. Starostenko, "HABE: Huffman Algorithm and Bit Extraction Applied to Image Equalization," in Electronics, Robotics and Automotive Mechanics Conference, Mexico, 2010.

[10] W. A. Finamore and M. B. d. Carvalho, "Lossy Lempel-Ziv on Subband Coding of Images," IEEE, Rio de Janairo, 1994.

[11] S. Ashida, H. Kakemizu, M. Nagahara and Y. Yamamoto, "Sampled-Data Audio Signal Compression with Huffman Algorithm," in SICE Annual Conference in Sapporo, Hokkaido, 2004.

[12] R. Zeimer and R. Peterson, Introduction to Digital Communication, New Jersey: Prentice Hall , 2001.

[13] O. Jalilian, A. Haghighat, and A. Rezvanian, "Evaluation of Persian text based on Huffman data compression," in XXII International Symposium on Information, Communication and Automation Technologies, 2009, pp. 1-5.

[14] G. Ong and S. Huang, "An efficient data compression scheme based on semi-adaptive Huffman coding for moderately large Chinese text files," in International Conference on Information Engineering, 1995, pp. 332-336.

[15] R. Weizheng, W. Haobo, X. Lianming, and C. Yansong, "Research on a quasi-lossless compression algorithm based on huffman coding," in International Conference on Transportation, Mechanical, and Electrical Engineering, 2011, pp. 1729-1732.

[16] N. Ranganathan and S. Henriques, "High-speed VLSI designs for Lempel-Ziv-based data compression," IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing, vol. 40, pp. 99-106, Feb. 1993.

[17] R. Das and T. Tuithung, "A novel steganography method for image based on Huffman Encoding," in Emerging Trends and Applications in Computer Science (NCETACS),3rd National Conference , Shillong, 2012.

[18] L. Flores-Pulido, M. C. L. Olivares-Gonzalez, M. Osorio and O. Starostenko, "HABE: Huffman Algorithm and Bit Extraction Applied to Image Equalization," in Electronics, Robotics and Automotive Mechanics Conference, Mexico, 2010.

[19] V. CmojeviC, V. Senk and Z. Trpovski, "Lossy Lempel-Ziv Algorithm," Telsiks, pp. 523-525, 1-3 October 2003.

[20] D. Kirovski and Z. Landau, "Generalized Lempel-Ziv Compression for Audio," in IEEE 6th Workshop on Multimedia Signal Processing, USA, 2004.

[21] D. I. Săcăleanu, R. Stoian and D. M. Ofrim, "An Adaptive Huffman Algorithm for Data Compression in Wireless Sensor Networks," IEEE, Bucharest, 2011.

[22] E.-H. Yang and J. Kieffer, "Simple Universal Lossy Data Compression Schemes Derived from the Lempel-Ziv Algorithm," IEEE TRANSACTIONS ON INFORMATION THEORY, vol. 42, no. 1, pp. 239 - 245, 1996.