*Communications on Applied Electronics (CAE) – ISSN : 2394-4714*
*Foundation of Computer Science FCS, New York, USA*
*Volume 7 – No. 7, October 2017 – www.caeaccess.org*

# Data Compression Methods and Analysis

Nazmun Nahar
Department of Computer Science & Engineering
Bangladesh, Dhaka

Md Jayedul Haque
Department of Computer Science & Engineering
Bangladesh, Dhaka

## ABSTRACT
During the advanced era of modern science data has become a salient part of research as well as other methodologies. Along with the cumulative use of data, data redundancy has become an ache for both user and researcher end. Not only communication but also generic file compression technologies are using different kind of efficient data compression methods massively day by day. This paper concerns with a variety of data compression methods with some efficient innovation. The purpose of data compression is to wan redundancy in stored or communicated data. Data compression has important application in the area of file storage and distributed system. This paper will provide an overture of several compression methods and will formulate new methods that may improve compression ratio and lessen error in the reconstructed data. In this work the data compression techniques: Huffman, Run-Length, LZW, LZW-Huffman, Huffman-LZW, Run-Length-LZW and LZW-Run-Length are used to compress different types of multimedia formats such as images and text, which depicts the discrepancy of various data compression methods on image and text file.

## General Terms
Computer Networks, Data Structure, Algorithm.

## Keywords
Lempel-Ziv-Welch (LZW), Huffman, Run-Length

## 1. INTRODUCTION
According to the nomenclature of data science, data compression is considered as a technique of converting an input data stream (the source stream or the original raw data) into another data stream (the output, the bit stream, or the compressed stream) that has a smaller size than before. A stream can be a file, a buffer in memory, or individual bits sent on a communications channel. The technique of reducing the size of a data file is popularly considered to as data compression. Compression has become effective because it helps to reduce resources usage, such as data storage space or transmission capacity. Since compressed data is decoded to use, this extra processing causes computational or other costs over decode. The sphere of data compression is often called source coding by the data scientist. The input symbols (such as bits, ASCII codes, bytes, audio samples, or pixel values) are emitted by a certain information source and have to be coded before being sent to their destination. There are two types of compression, lossy and lossless. Lossy compression reduced file size by abrogating some unneeded data that won't be recognize by human after decoding, this often used by video and audio compression. On the other hand, lossless compression manipulates each bit of data inside file to minimize the size without losing any data after decoding.

## 2. LITERATURE REVIEW
In 1952 Huffman [10] coined an elegant sequential algorithm which generates optimal prefix codes in O (nlogn) time. The algorithm in fact requires only linear time providing that the frequencies of appearances are sorted in advance. There have been comprehensive researches on analysis, implementation issues and improvements of the Huffman coding theory in a variety of applications [11, 12]. In [13], they analyze one of the coding techniques on OFDM, named Huffman coded OFDM (HC-OFDM) that contributes not only to high data rate but also prevents peak the signals when they sum up after IFFT process, while decreasing the BER at 10-1. HC-OFDM simulations reached valuable results on a big data stream transmission and lower peak to average power ratio (PAPR) probability over Rayleigh fading channel and phase offset comparison at the detection which their BER and PAPR values are smaller at and 3-4 db respectively than an uncoded OFDM. A new multimedia functional unit for general-purpose processors has been proposed [14] in order to increase the performance of Huffman coding. After that In 1984 LZW introduced a new compression technique. One of the lossless data compression widely used is LZW data compression, it is a dictionary based algorithm. LZW compression is named after its developers, A. Lempel and J. Ziv, with later modifications by Terry A. Welch [1]. Lempel-Ziv-Welch (LZW) [1] this algorithm proposed by Welch in 1984. LZW compression works best for files containing lots of repetitive data. This is often the case with text as well as monochrome images. LZW compression is fast comparing to other algorithms. This algorithm is an improved implementation of the LZ78 algorithm published by Lempel and Ziv in 1978 (LZ78) [2]. The first algorithm of Lempel and Ziv was published in 1977 and it is named as LZ77 [3]. LZ-77 is an example of what is known as "substitutional coding". The LZ77 [3] and LZ78 [2] are otherwise called LZ1 and LZ2 respectively like The LZW algorithm uses dictionary and index for encoding and decoding operation. In 2011 Senthil Shanmugasundaram and Robert Lourdusamy[15] worked on Statistical compression techniques and Dictionary based compression techniques which was performed on text data. In between the statistical coding techniques the algorithms such as Shannon-Fano Coding, Huffman coding, Adaptive Huffman coding, Run Length Encoding and Arithmetic coding were considered in his research. Lempel Ziv scheme which is a dictionary based technique was divided into two families: those derived from LZ77 (LZ77, LZSS, LZH and LZB) and those derived from LZ78 (LZ78, LZW and LZFG) in his work. After a short period, in 2013, Doa'a Saad El-Shora & Ehab Rushdy Mohamed works on the data compression techniques: Huffman, Adaptive Huffman and arithmetic, LZ77, LZW, LZSS, LZHUF, LZARI and PPM are tested against different types of data with different sizes. In 2014, Kashfia Sailunaz, Mohammed Rokibul Alam Kotwal worked on Shannon Fano Coding, Huffman Coding, Repeated Huffman Coding and Run-Length coding. A new algorithm "Modified Run-Length Coding" is also proposed and compared with the other algorithms only on full text data. It

was a great contribution in the field of data compression. After a while in 2017, Jayedul Haque and Nurul Huda worked on Huffman, Run-Length, LZW, Shannon-Fano, Repeated-Huffman, Run-Length-Huffman, and Huffman-Run-Length [19] which were tested against not only text file but also image file. This paper has extended their work using both image and text data considering their future implementation.

# 3. METHOD OF DATA COMPRESSION

## 3.1 Recent Compression Method

### *3.1.1 Lempel-Ziv-Welch (LZW)*

In the introduction chapter it has been discussed that what lossless data compression is. One of the lossless data compression widely used is LZW data compression, it is a dictionary based algorithm. LZW compression is named after its developers, A. Lempel and J. Ziv, with later modifications by Terry A. Welch [1]. Lempel-Ziv-Welch (LZW) [1] this algorithm proposed by Welch in 1984. LZW compression works best for files containing lots of repetitive data. This is often the case with text and monochrome images. LZW compression is fast comparing to other algorithms. This algorithm is an improved implementation of the LZ78 algorithm published by Lempel and Ziv in 1978 (LZ78) [2]. The first algorithm of Lempel and Ziv was published in 1977 and it is named as LZ77 [3]. LZ-77 is an example of what is known as "substitutional coding". The LZ77 [3] and LZ78 [2] are otherwise called LZ1 and LZ2 respectively like The LZW algorithm uses dictionary and index for encoding and decoding operation. It creates a dictionary and if a match is found in the dictionary then corresponding string is replaced by the index. LZW compression became the first widely used universal data compression method on computers. After the invention of LZW there are lots of improvements and enhancement done in LZW for data compression that is discussed in this section. LZW compression works best for files containing lots of repetitive data especially for text and monochrome images.

### *3.1.2 Run-Length*

Run-Length Encoding is considered the easiest method of compression techniques which can be used to compress data made of any combination of symbols. It does not need to know the frequency of repetition of symbols and can be very efficient if data is represented as 0s and 1s [4].

The common technique behind this algorithm is to replace consecutive reiterating occurrences of a symbol by one occurrence of the symbol followed by the number of occurrences.

### *3.1.3 Huffman*

Huffman codes which are optimal with prefix codes generated from a set of probabilities by a particular algorithm, the Huffman Coding Algorithm. David Huffman developed the algorithm as a student in a class on information theory at MIT in 1950. The algorithm is now probably the most prevalently used component of compression algorithms, used as the back end of GZIP, JPEG and many other utilities.[5][6][7][8]

The basic understanding of Huffman coding is a technique which deals with data compression of ASCII characters. It follows top down procedure means the binary tree is built from the top down to construct a minimal consequence. In Huffman Coding the characters in a data file are converted to binary code and the most common characters in the file have the shortest binary codes, and the characters which are least common have the longest binary code [9].

# 4. INTRODUCED METHODS

## 4.1 LZW-Huffman

Both the Huffman and LZW is an effective methods for data compression techniques. LZW is better for text and monochrome image where Huffman gives better result for all the multimedia file. If Huffman coding technique can be applied effectively on a file after LZW method, then it is called LZW-Huffman coding. An algorithm which compresses data using LZW encoding then it uses Huffman encoding on resultant data.

Algorithm 1 illustrates how LZW-Huffman coding works.

Step 1: At the start, the dictionary and P is empty;
Step 2: C = next character in the input-stream;
Step 3: Is the string P+C present in the dictionary?

- if
  yes: P = P+C (extend P with C);
- if not:
  - output the code word which denotes P to the output-stream;
  - add the string P+C to the dictionary;
  - P = C (P now contains only the character C);

Step 4: Are there more characters in the input-stream?

- if yes: go back to step 2;
- if not:
  - output the code word which denotes P to the output-stream;

Step 5: Scan output stream
Step 6: for (Read byte start to end)
{
Found-bytes [index] =read byte;
Frequency [index] =count the repetition of byte; //filling frequency table
}
Step 7: Build Sorted Frequency Table
Step 8: Build Huffman Tree according to table
Step 9: Traversal of tree to determine all code words in bits
Step 10: Make byte-list reading this bit-list
Step 11: Serialize byte-list in a file with extension *.jna in browsing directory.

**Algorithm 1: LZW-Huffman Coding**

## 4.2 Huffman-LZW

Both the Huffman and LZW is an effective methods for data compression techniques. LZW is better for text and monochrome image where Huffman gives better result for all the multimedia file. If Huffman coding technique can be applied effectively on a file then LZW method, then it is called Huffman-LZW coding. An algorithm which compresses data using Huffman encoding then it uses LZW encoding on resultant data. .

Algorithm 2 illustrates how Huffman-LZW coding works.

Step 1: Scan input stream
Step 2: for (Read byte start to end)
{
Found-bytes [index] =read byte;
Frequency [index] =count the repetition of byte; //filling frequency table
}
Step 3: Build Sorted Frequency Table
Step 4: Build Huffman Tree according to table
Step 5: Traversal of tree to determine all code words in bits
Step 6: Make byte-list reading this bit-list
Step 7: At the start, the dictionary and P is empty;
Step 8: C = next character in the input-stream;
Step 9: Is the string P+C present in the dictionary?

- if
  yes: P = P+C (extend P with C);
- if not:
  - output the code word which denotes P to the output-stream;
  - add the string P+C to the dictionary;
  - P = C (P now contains only the character C);

Step 10: Are there more characters in the input-stream?

- if yes: go back to step 2;
- if not:
  - output the code word which denotes P to the output-stream;

Step 11: Make byte-list
Step 12: Serialize byte-list in a file with extension *.jna in browsing directory.

**Algorithm 2: Huffman-LZW Coding**

## 4.3 LZW-Run-Length

If LZW coding technique can be applied effectively on a file before Run-length algorithm, then it is called LZW-Run-length coding. An algorithm which compress data using LZW encoding then it uses Run Length encoding on resultant data. First it builds dictionary on scanned file after that it traverse dictionary to make the code and detects repeating occurrences. After that Run-Length is applied for the further procedure.

Algorithm 3 illustrates how LZW-Run-length coding works.

Step 1: At the start, the dictionary and P is empty;
Step 2: C = next character in the input-stream;
Step 3: Is the string P+C present in the dictionary?

- if
  yes: P = P+C (extend P with C);
- if not:
  - output the code word which denotes P to the output-stream;
  - add the string P+C to the dictionary;
  - P = C (P now contains only the character C);

Step 4: Are there more characters in the input-stream?

- if yes: go back to step 2;
- if not:
  - output the code word which denotes P to the output-stream;

Step 5: Make byte-list reading this output stream
Step 6: Scan byte-list
Step 7: Replace consecutive repeating occurrences
Step 8: Insert symbol with occurrence in binary array
Step 9: Form byte array from binary array
Step 10: Serialize byte array in a file with extension *.jts in browsing directory

**Algorithm 3: LZW-Run-length Coding**

## 4.4 Run-Length-LZW

If Run-Length coding technique can be applied effectively on a file before LZW algorithm, then it is called Run-length-LZW coding. An algorithm which compress data using Run-Length encoding then it uses LZW encoding on resultant data. First it detects repeating occurrences on scanned file then builds dictionary using this repeating occurrences after that it traverse dictionary to make the code.

Algorithm 4 illustrates how Run-length-LZW coding works.

Step 1: Scan file from browsing directory
Step 2: Replace consecutive repeating occurrences
Step 3: Insert symbol with occurrence in byte list
Step 4: Scan byte list as input stream
Step 5: At the start, the dictionary and P is empty;
Step 6: C = next character in the input-stream;
Step 7: Is the string P+C present in the dictionary?

- if
  yes: P = P+C (extend P with C);
- if not:
  - output the code word which denotes P to the output-stream;
  - add the string P+C to the dictionary;
  - P = C (P now contains only the character C);

Step 8: Are there more characters in the input-stream?

- if yes: go back to step 2;
- if not:
  - output the code word which denotes P to the output-stream;

Step 9: Make byte-list
Step 10: Serialize byte array in a file with extension *.jna in browsing directory

**Algorithm 4: Run-length-LZW Coding**

*Communications on Applied Electronics (CAE) – ISSN : 2394-4714*
*Foundation of Computer Science FCS, New York, USA*
*Volume 7 – No. 7, October 2017 – www.caeaccess.org*

## 5. RESULT ANALYSIS

This paper will establish the effectiveness of LZW-Huffman coding, Huffman-LZW, Run-length-LZW and LZW-Run-Length algorithm. Text and image file has been used to test those compression methods. We executed and tested our methods on many standard and famous images such as "Lena image" and other famous images. These standard test images have been used by different researchers [16-20] related to image compression and image applications. We have used 256×256 image file size. For assessing effectiveness of methods compression ratio is used. In addition a fraction of enwik8 text [21] is used in our work to evaluate the compression techniques. As enwik8 is a large file so that to avoid time complexity we have used a smaller part of this file to analyze the result. Compression ratio is defined as

$$\text{Compression ratio} = \frac{Original - Compressed}{Original} \times 100$$

File containing Huffman tree has the format that is discussed in [8] Repeated Huffman coding was first used with normal coding of the tree and then memory efficient coding was used to see whether repetition count increases. A Huffman tree representation is also related to average code length for a symbol in a message.

## 6. CAPTIONS/FIGURES

In this work all (previously mentioned in abstract) algorithms have implemented using net-beans. Table 1 illustrate that all data compression techniques achieve negative and positive results against standard files. Positive results mean it lessen the file size and negative results mean it increases the file size. The experimental results of the implemented algorithms, Huffman, Run-Length, LZW coding as well as proposed methods LZW-Huffman, Run-Length-LZW, Huffman-LZW and LZW-Run-Length for compression ratio are described in Table 1.

**Table 1. Compression ratio for images and text data in compression methods**

| File | Original Size(Bytes) | Compression Ratio (%) | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | Run-length | Huffman | LZW | LZW-Huffman | Huffman-LZW | LZW-Run-length | Run-length-LZW |
| lena.jpg | 8,246 | -44.3 | 0.74 | 0.9 | 7.8 | -2.5 | -34 | 0.98 |
| F16.jpg | 8,628 | -32.2 | 0.54 | 0.7 | 8.2 | 1.6 | -72 | -1.8 |
| babbon.jpg | 11,655 | -45.3 | 1 | -0.98 | 11.2 | 3.5 | -89 | 1.6 |
| boat.jpg | 20,561 | -48.7 | 1 | 0.6 | 9.6 | -13.1 | -158 | -23 |
| Peppers.jpg | 8,690 | -34.1 | 0.44 | 1.3 | 6.9 | 11.4 | -289 | 1.7 |
| Enwik8.txt | 16,47,84 | -0.9 | 37.1 | 44.5 | 53 | 39.2 | 28 | 1.4 |

The apparent of Huffman is its compression ratio is always positive for all sample file. On the other hand, LZW methodology also gives positive ratio except one. From these table we can see that Huffman is giving better result for image file. On the other hand, LZW method is better for text data. Again we can see Run-Length method is not efficient for both image and text document as it gives negative result for all the sample file.

The compression ratio of LZW-Huffman and Huffman-LZW are better than other two proposed technique comparatively. But only LZW-Huffman gives some positive results for standard image file and text file.

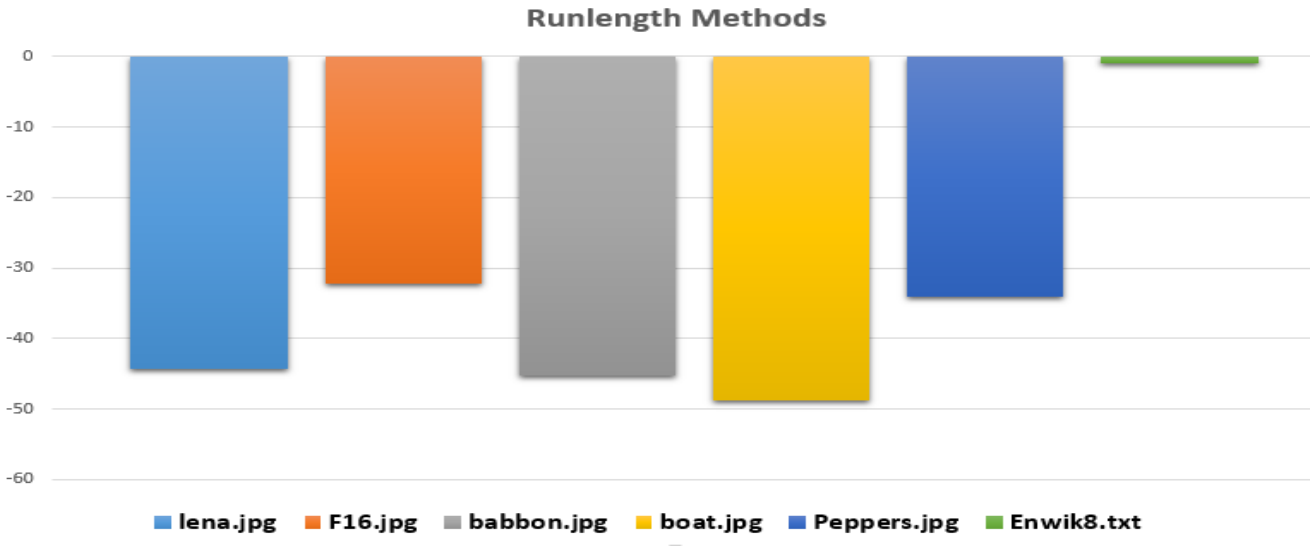


**Fig 1: Results for Run-Length**

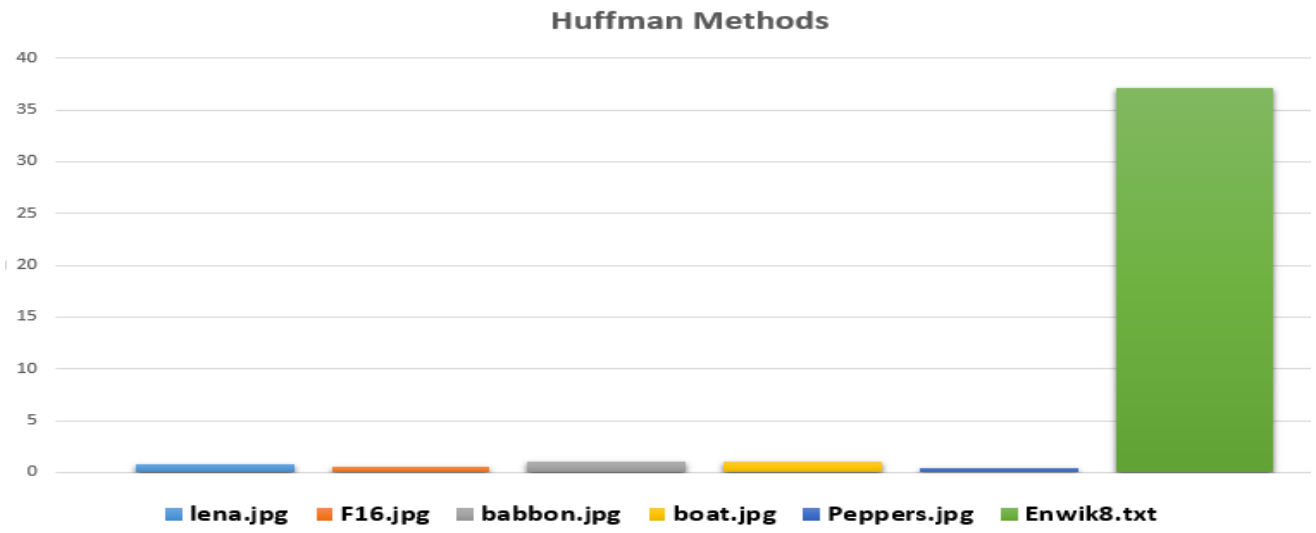


**Fig 2: Results for Huffman**

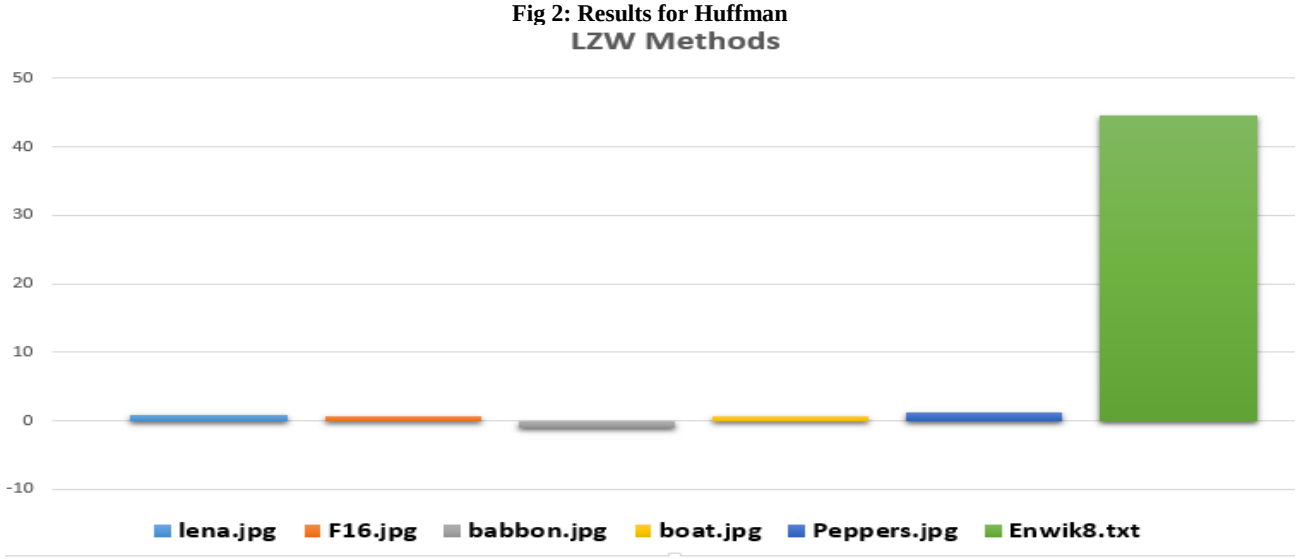


**Fig 3: Results for LZW**

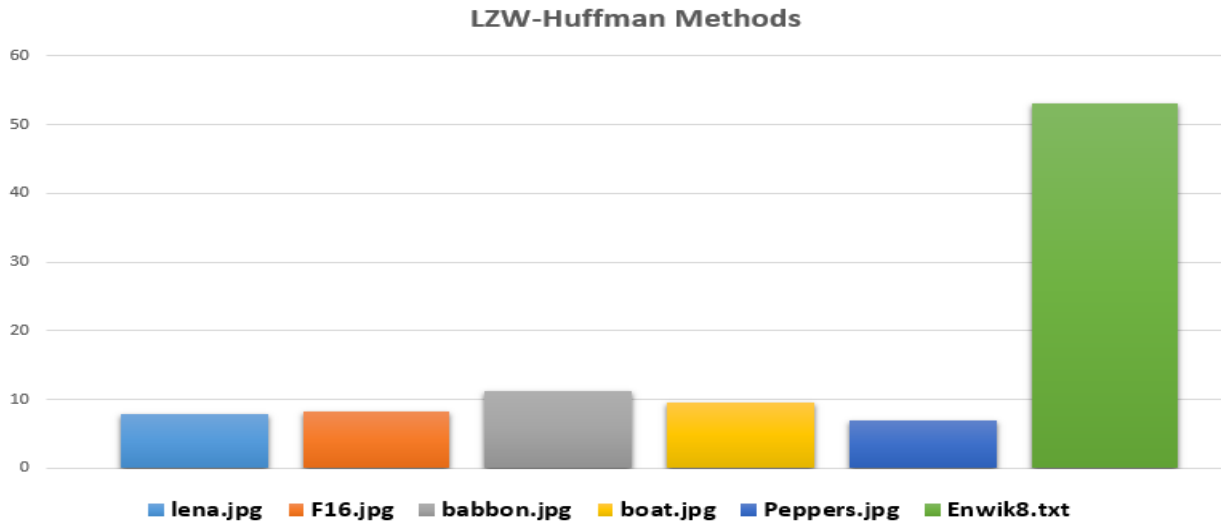


**Fig 4: Results for LZW-Huffman**

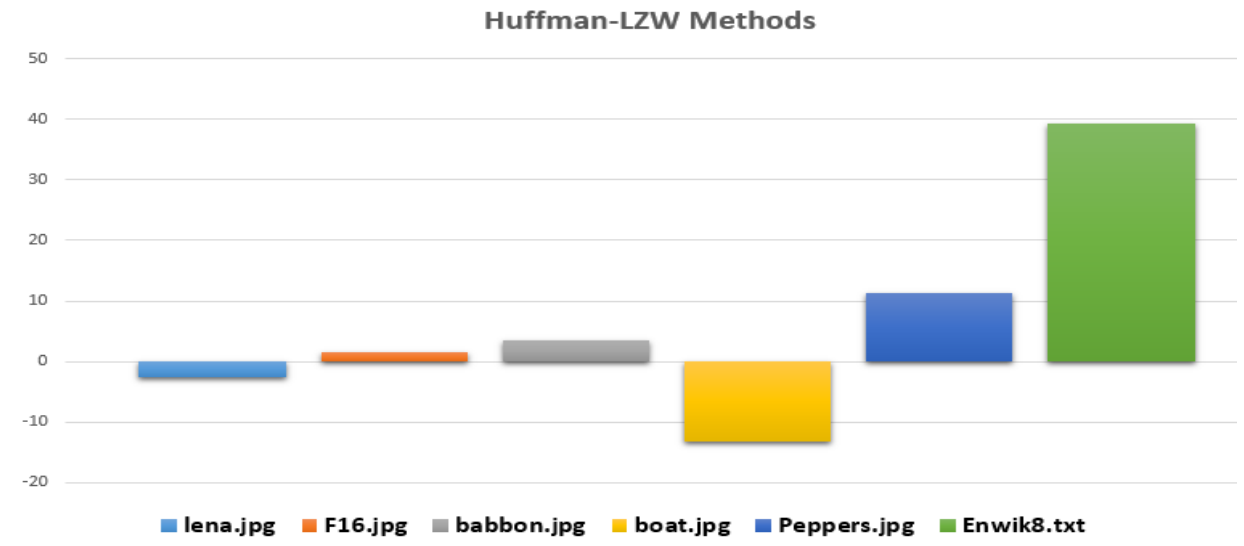


**Fig 5: Results for Huffman-LZW**

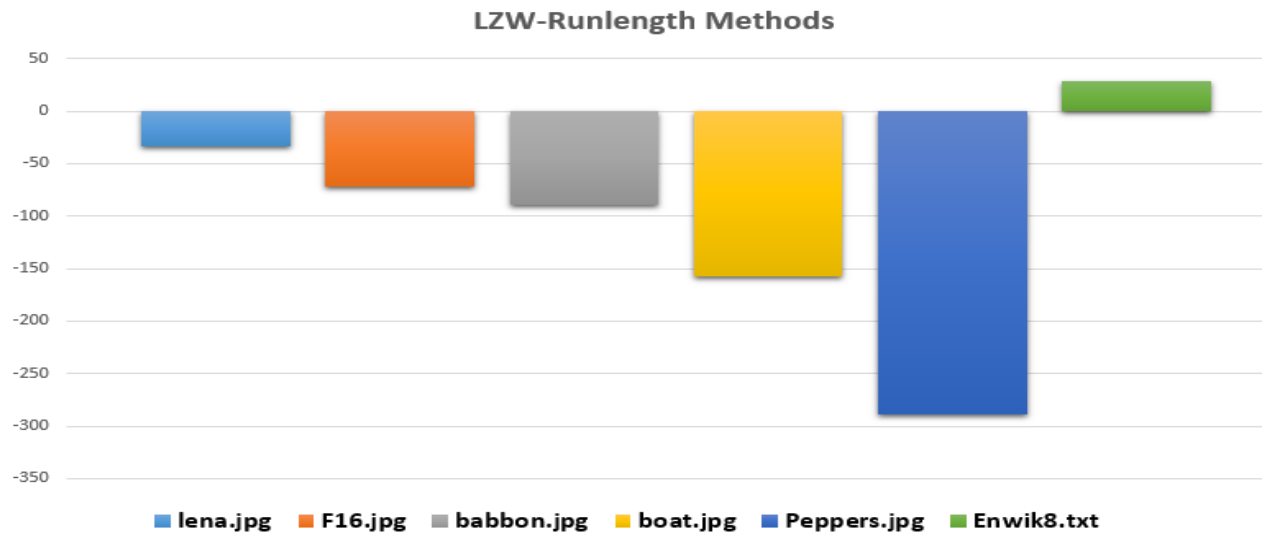


**Fig 6: Results for LZW-Run-Length**

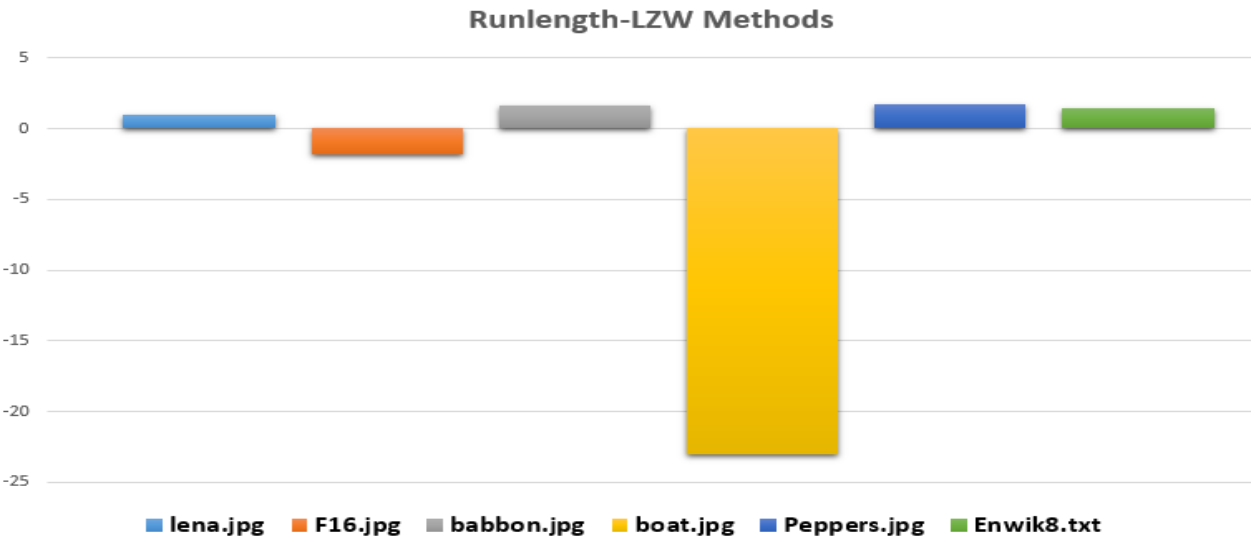


**Fig 7: Results for Run-Length-LZW**

There are seven figures given above based on all of our methods which are used to compress benchmark files as we have mentioned in result analysis chapter. These figures represents the effectiveness of each method which was experienced by the benchmark files. If we look into the fig 1-fig 4, fig 1 have opposite eminence regarding rest. Run-length have negative compression ratio on the other hand Huffman, LZW and LZW-Huffman have positive ratio for all the benchmark file except LZW for babbon.jpg. And the other three proposed methods have mishmash of ratio that means some method have positive ratio and some have negative for the respective benchmark files. Eventually text file is positively compressed by all of the methods except runlegth.

# 7. CONCLUSION

Data compression is a way that decreases the data size, removing the obsessive information and redundancy which consequently reduces the storage space, cost & increases the data transfer rate in communication. In this paper recent methods are associated in a single method. The identical features of these algorithms are transformation and compression algorithms; where the transformation rearranged the data to optimize input for the next sequence of compression algorithm. Those proposed methods are experimented with different benchmark files and formats such as images and text files which represents some prospective results with few drawbacks in proposed methods. The comparison of Experiment results with the recent methods and proposed algorithms hits the expected better compression ratio (%) for "LZW-Huffman", which gives better result even from Huffman for text data as well as image. On the other hand LZW-Run-Length gives a better result in comparison with other proposed method for only text data. But Huffman gives better result for all the image and text files which can be seen in table given in figures and tables section. The main drawback experienced in this paper is the compression ratio (%) of "LZW- Run-length" algorithm, which only manages a Compression ratio (%) in negative magnitude for image file. Eventually, it remains space for the future research and development on several fields which can be carried on like Repeated LZW, Repeated Run-Length and Repeated Huffman-LZW compression techniques. Besides, compression coding video and audio data and efficient decoding technique for all the proposed methods will be carried on in future works.

# 8. ACKNOWLEDGMENTS

Our gladness to Almighty and my supervisor for his relentless conduct, encouragement, and endurance, and for giving me the opportunity to do this work.

Eventually, my profound gratefulness and predilection to my parents for their buttress, encouragement, and incessant love.

# 9. REFERENCES

[1] WELCH, T. A. 1984." A technique for high-performance data compression". IEEE Comput. 17, 6, 8–19. 9.

[2] ZIV, J. AND LEMPEL, A. 1978. "Compression of individual sequences via variable-rate coding". IEEE Trans. Inform. Theory 24, 5, 530–536.

[3] ZIV, J. AND LEMPEL, A. 1977. A "universal algorithm for sequential data compression". IEEE Trans. Inform. Theory 23, 3, 337–343.

[4] Campos, A. S. E. Run Length Encoding. Available: http://www.arturocampos.com/ac_rle.html (last accessed July 2012).

[5] Connel, J. B., "A Huffman-Shannon-Fano Code", Proc. IEEE 61 (Jul. 1973), 1046-1047.

[6] Gallager, R. G., "Variations on a theme by Huffman", IEEE Trans. Inf. Theory IT-24, 6(Nov. 1978), 668-674.

[7] Hashemian, R., "Memory efficient and high-speed search Huffman coding", IEEE Trans. Comm. 43(10)(1995)2576-2581.

[8] M. N. Huda, "Study on Huffman Coding," Graduate Thesis, 2004.

[9] 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.

[10] Huffman, D. A. : 'A Method for the Construction of Minimum Redundancy Codes", Proc. IRE, Vol. 40, No. 9, pp. 1098-1101, September 1952.

[11] Buro. M.: 'On the maximum length of Huffman codes', Information Processing Letters, Vol. 45, No.5, pp. 219-223, April 1993.

[12] Chen, H. C. and Wang, Y. L. and Lan, Y. F.: 'A Memory Efficient and Fast Huffman Decoding Algorithm'Information Processing Letters, Vol. 69, No. 3, pp. 119- 122, February 1999.

[13] Mashhur Sattorov, Heau-Jo Kang. :' Huffman Coding Approach to Performance of 16-QAM/OFDM'.

[14] Wong, S. and Cotofana, D. and Vassiliadis, S.: General-Purpose Processor Huffman Encoding Extension, Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC 2000), pp. 158-163, Las Vegas, Nevada, March 2000.

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

[16] Kao, Ch., H, and Hwang, R. J.: 'Information Hiding in Lossy Compression Gray Scale Image', Tamkang Journal of Science and Engineering, Vol. 8, No 2, 2005, pp. 99-108.

[17] Ueno, H., and Morikawa, Y.: 'A New Distribution Modeling for Lossless Image Coding Using MMAE Predictors'. The 6th International Conference on Information Technology and Applications, 2009.

[18] Grgic, S., Mrak, M., and Grgic, M.: 'Comparison of JPEG Image Coders'. University of Zagreb, Faculty of Electrical Engineering and Computing Unska 3 / XII, HR-10000 Zagreb, Croatia.

[19] Md Jayedul Haque and Mohammad Nurul Huda. Study on Data Compression Technique. *International Journal of Computer Applications* 159(5):6-13, February 2017.

[20] http://sipi.usc.edu, accessed Mar 2011.

[21] Fano R.M., "The Transmission of Information", Technical Report No. 65, Research Laboratory of Electronics, M.I.T., Cambridge, Mass.; 1949.