International Journal of Computer Applications (0975 – 8887)
Volume 159 – No 5, February 2017

Study on Data Compression Technique

Md. Jayedul Haque
Department of Computer
Science & Engineering
United International University
Bangladesh, Dhaka-1100

Mohammad Nurul Huda
Professor & MSCSE Coordinator
United International University
Bangladesh, Dhaka

ABSTRACT

In this current age both communication and generic file compression technologies are using different kind of efficient data compression methods massively. This paper surveys a variety of data compression methods. The aim of data compression is to reduce 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 overview of several compression methods and will formulate new algorithms that may improve compression ratio and abate error in the reconstructed data. In this work the data compression techniques: Huffman, Run-Length, LZW, Shannon-Fano, Repeated-Huffman, Run-Length-Huffman, and Huffman-Run-Length are tested against different types of multimedia formats such as images and text, which shows the difference of various data compression methods on image and text file.

General Terms

Network Security, Algorithms, Hidden message in DNA, File System, Modems and Files (BOA)

Keywords

Lempel-Ziv-Welch (LZW), Huffman, Shannon-Fano, Data Compression, Benchmark file, Data Structure, Algorithms

1. INTRODUCTION

Data compression is the process 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. A stream can be a file, a buffer in memory, or individual bits sent on a communications channel. The process of reducing the size of a data file is popularly referred to as data compression. Compression is useful because it helps to reduce resources usage, such as data storage space or transmission capacity. As compressed data must be decompressed to use, this extra processing imposes computational or other costs through decompression. The field of data compression is often called source coding. 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 1949 Shannon – Fano algorithm was simultaneously developed by Claude Shannon (Bell labs) and R.M. Fano (MIT) [16]. It is used to encode messages depending upon their probabilities. After a short period, Huffman [21] in 1952 proposed an elegant sequential algorithm which generates optimal prefix codes in O (nlogn) time. The algorithm actually needs only linear time provided that the frequencies of appearances are sorted in advance. There have been extensive researches on analysis, implementation issues and improvements of the Huffman coding theory in a variety of applications [17, 18]. In [19], a two-phase parallel algorithm for time efficient construction of Huffman codes has been proposed. A new multimedia functional unit for general-purpose processors has been proposed [20] 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 [7]. Lempel-Ziv-Welch (LZW) [7] 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) [8]. The first algorithm of Lempel and Ziv was published in 1977 and it is named as LZ77 [9]. LZ-77 is an example of what is known as "substitutional coding". The LZ77 [9] and LZ78 [8] 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[10] 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. This paper has extended their work using both image and text data in comparison.

3. DATA COMPRESSION

3.1 Recent Compression Technique

3.1.1 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.[1][2][3][4]

Huffman coding 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 [5].

3.1.2 Run-Length

Run-Length Encoding might be the simplest 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 [6].

The general idea behind this method is to replace consecutive repeating occurrences of a symbol by one occurrence of the symbol followed by the number of occurrences.

3.1.3 Lempel-Ziv-Welch (LZW)

In the introduction chapter we have discussed about the lossless data compression. 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 [7]. Lempel-Ziv-Welch (LZW) [7] 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) [8]. The first algorithm of Lempel and Ziv was published in 1977 and it is named as LZ77 [9]. LZ-77 is an example of what is known as "substitutional coding". The LZ77 [9] and LZ78 [8] 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. There are several algorithms like DEFLATE and GZIP uses the LZ family algorithms. 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. In this work we have used LZW only for text data.

3.1.4 Shannon-Fano Coding

This method is called an earliest technique for data compression that was invented by Claude Shannon and Robert Fano [10] in 1949. It is used to encode data depending upon their probabilities. The algorithm for Shannon- Fano coding is:

1. According to given list of ASCII characters, build a frequency or probability table.
2. Sort out the table according to the frequency, with the most frequently occurring character on the top.
3. Divide the table into two halves with the total frequency count of the upper half being as close to the total frequency count of the bottom half as possible.
4. Assign the upper half of the list a binary digit ‘0’ and the lower half a ‘1’.
5. Repeatedly apply the steps 3 and 4 to each of the two halves, subdividing groups and adding bits to the codes until each symbol has become a corresponding leaf on the tree.

Generally, Shannon-Fano coding does not guarantee the generation of an optimal code. Shannon – Fano algorithm is more efficient when the probabilities are closer to inverses of powers of 2 [10].

3.2 Usage of Data Compression

The usage of data compression are massive. We use data compression when to preserve information or file in safe manner. Besides, if we are uploading one or several large files to a website or moving files to another machine, you can get benefit from a smaller file size. Alongside, Email servers are notoriously stringent on the size of the file attachments that you can have. Compressing the files will reduce the file size and allow you to send more files into the message as an attachment.

4. PROPOSED METHODS

4.1 Repeated Huffman

If Huffman coding technique can be applied effectively on a file again and again, then it is called Repeated Huffman coding. An algorithm which compresses data using Huffman encoding then it again uses Huffman encoding on resultant data. The Huffman tree is built twice the time and traversal process is same to build code. While it is expected that encoded message length will be smaller in every pass of Repeated Huffman coding, nevertheless encoding the tree itself will be an overhead in each pass. So repetition count will depend upon how efficiently we can represent a Huffman tree. If a Huffman tree can be represented efficiently in memory, Repeated Huffman coding technique can be applied in an effective number of times.

Algorithm 1 illustrates how Repeated Huffman coding works.

```
Step 1:  Scan file from browsing directory
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:  Scan byte-list
Step 8:  Read byte to again make Frequency Table go- to Step2
Step 9:  go to Step3
Step 10:  go to Step4
Step 11:  go to Step5
Step 12:  go to Step6
Step 13:  Serialize byte-list in a file with extension *.jts in browsing directory.
```

Algorithm 1: Repeated Huffman Coding

4.2 Run-Length Huffman

If Huffman coding technique can be applied effectively on a file after Run-length algorithm, then it is called Run-length Huffman coding. An algorithm which compresses data using Run Length encoding then it uses Huffman encoding on resultant data. First it detects repeating occurrences and build Huffman tree on optimal characters after that it traverse tree to make the code. While it is expected that encoded message length will be smaller in every pass of Run-length Huffman coding.

Algorithm 3 illustrates how Repeated Huffman 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
Step 5:   for (Read byte start to end)
          {
                Found-bytes [index] =read byte;
                Frequency [index] =count the repetition of byte; //filling frequency table
          }
Step 6:   Build Sorted Frequency Table
Step 7:   Build Huffman Tree
Step 8:   Traversal of tree to determine all code words in bits
Step 9:   Make byte list reading this bit list
Step 10:  Serialize byte list in a file with extension *.jts in browsing directory
```

Algorithm 2: Run-length Huffman Coding

4.3 Huffman Run-Length

If Huffman coding technique can be applied effectively on a file before Run-length algorithm, then it is called Huffman Run-length coding. An algorithm which compress data using Huffman encoding then it uses Run Length encoding on resultant data. First it builds Huffman tree on scanned file after that it traverse tree to make the code and detects repeating occurrences. While it is expected that encoded message length will be smaller in every pass of Huffman Run-length coding.

Algorithm 2 illustrates how Huffman Run-length coding works.

```
Step 1:   Scan file from browsing directory
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:   Scan byte-list

Step 8:   Replace consecutive repeating occurrences

Step 9:   Insert symbol with occurrence in binary array

Step 10:  Form byte array from binary array

Step 11:  Serialize byte array in a file with extension *.jts in browsing directory
```

Algorithm 3: Huffman Run-length Coding

5. RESULTS AND DISCUSSION

This paper will establish the effectiveness of Repeated Huffman coding, Huffman Run-length, Run-length Huffman and experimental results of these three 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 [11-14] related to image compression and image applications. We have used 256×256 image file size. For assessing effectiveness of methods compression ratio is used. Besides, pg571 text file have been used [15] by so many researchers in there research. In addition a fraction of enwik8 text [16] 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 [4] 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. Average code length can be defined as

$$\text{Average code length, AL}= \sum_{i=1}^{n} p_i l_i$$

Where pi is the probability of ith symbol

And li is the code-length of ith symbol.

5.1 Figures and Tables

In this work all (previously mentioned in abstract) algorithms have implemented using visual studio 2012. 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 and Shannon Fano coding as well as proposed methods Repeated Huffman, Run-Length Huffman and Huffman 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*	*Shannon-Fano*	*Repeated Huffman*	*Huffman Run-length*	*Run-length Huffman*
lena.jpg	8,246	-56.3	0.39	N/A	-469	-36.13	-346	0.7
F16.jpg	8,628	-58.2	0.40	N/A	-469	-35.2	-772	-1.5
babbon.jpg	11,655	-62.3	1	N/A	-467	-25.3	-989	2.3
boat.jpg	20,561	-49.7	1	N/A	-465	-13.1	-958	13.9
Peppers.jpg	8,690	-54.1	0.38	N/A	-469	-34.6	-2849	1.9
Pg571.txt	3,013,373	-93.8	35.13	41	-317	36.3	-224.24	13.4
Enwik8.txt	16,47,842	-2.9	36.1	44.5	-364	37.2	-180	14.4

The significance of Huffman is its compression ratio is always positive for all sample file. On the other hand, LZW methodology only implemented for text documents. From these two table we can see only Huffman is giving better result for image file. On the other hand, LZW method is better for text data. But, Shannon Fano is not giving any good result for any file format. Again we can see Run-Length method is not efficient for image compression as it also gives negative result.

The compression ratio of Repeated Huffman and Huffman Run-Length is not positive for all sample image file in this assumption. Only Run-Length Huffman gives some positive results for standard image file.

Again, we can see the result of Repeated Huffman is the best for text data where Huffman Run-Length is always negative. Not only Repeated Huffman but also Run-Length Huffman gives positive result for text data.

In fig 1 – fig 7 we have shown all the results of compression methods with each standard files in histogram plot separately. Which will be helpful for a reader to get a bird's eye of view of all the compression methods in a short time. From experimental results we can make a summary that Huffman and Run-Length Huffman gives us the better compression result comparison with other methods. After using all of the summary we have made a relationship chart to understand that which benchmark file is giving better and efficient result for all of the methods. Where we can easily make a conclusion that enwik8 text is best one for all the compression techniques

.

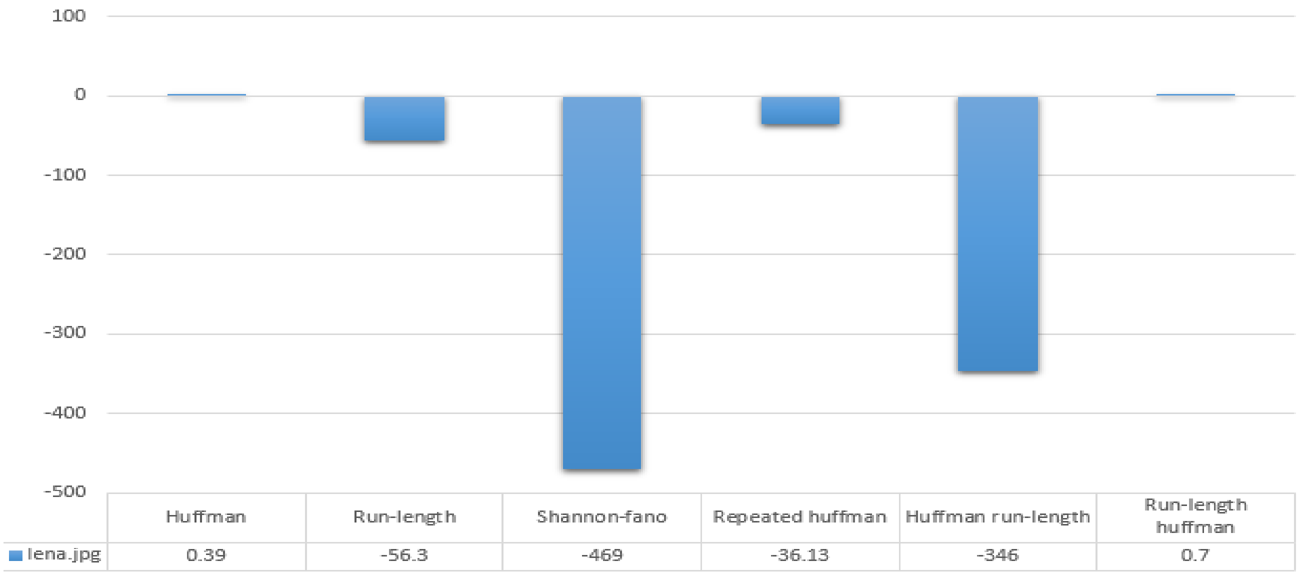
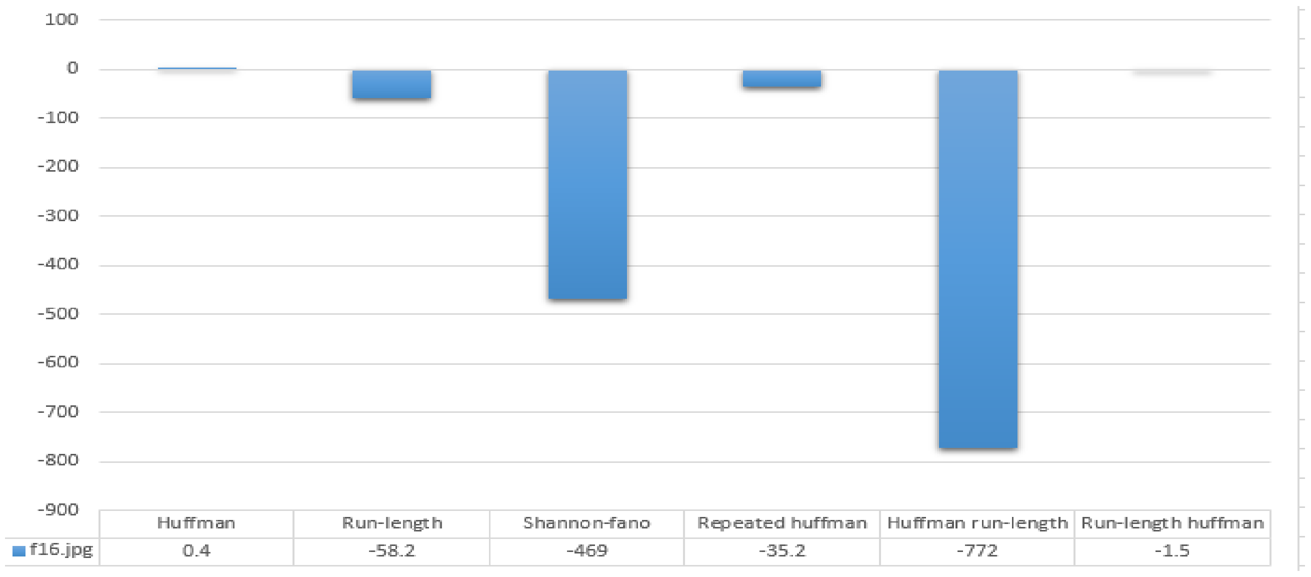
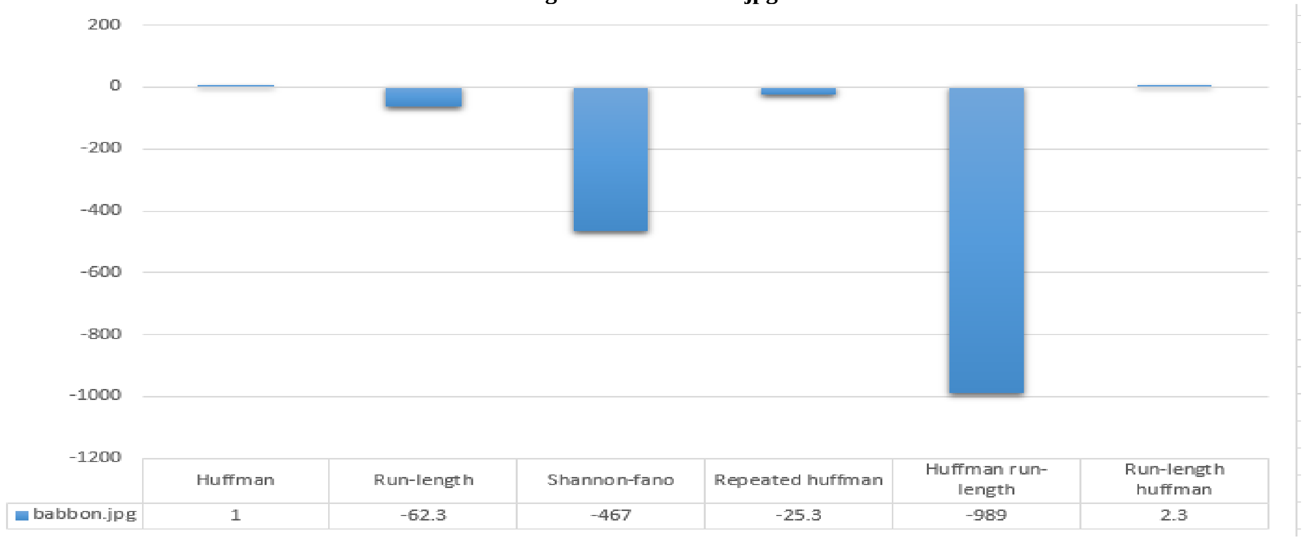


Fig 1: Results for lena.jpg



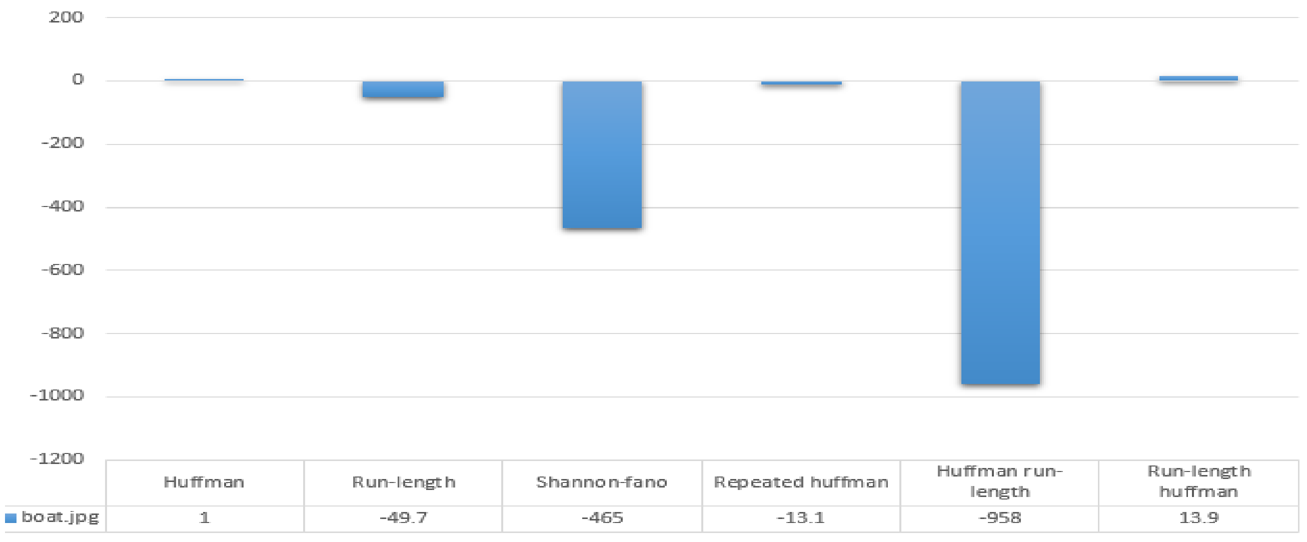
	Huffman	Run-length	Shannon-fano	Repeated huffman	Huffman run-length	Run-length huffman
🟦 f16.jpg	0.4	-58.2	-469	-35.2	-772	-1.5

Fig 2: Results for f16.jpg



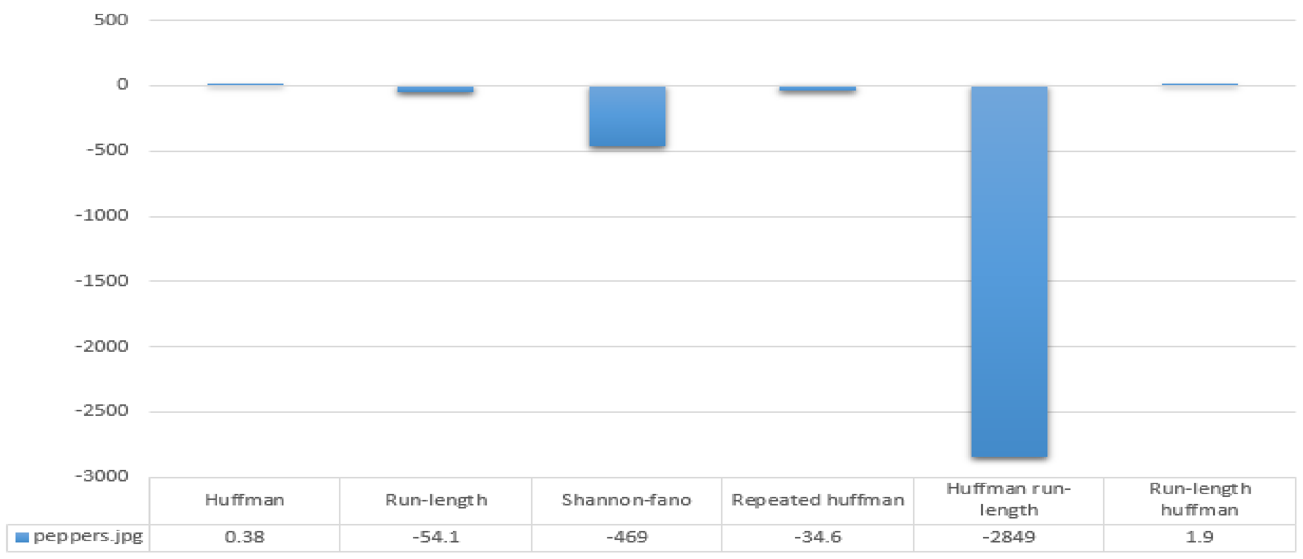
	Huffman	Run-length	Shannon-fano	Repeated huffman	Huffman run-length	Run-length huffman
🟦 babbon.jpg	1	-62.3	-467	-25.3	-989	2.3

Fig 3: Results for babbon.jpg



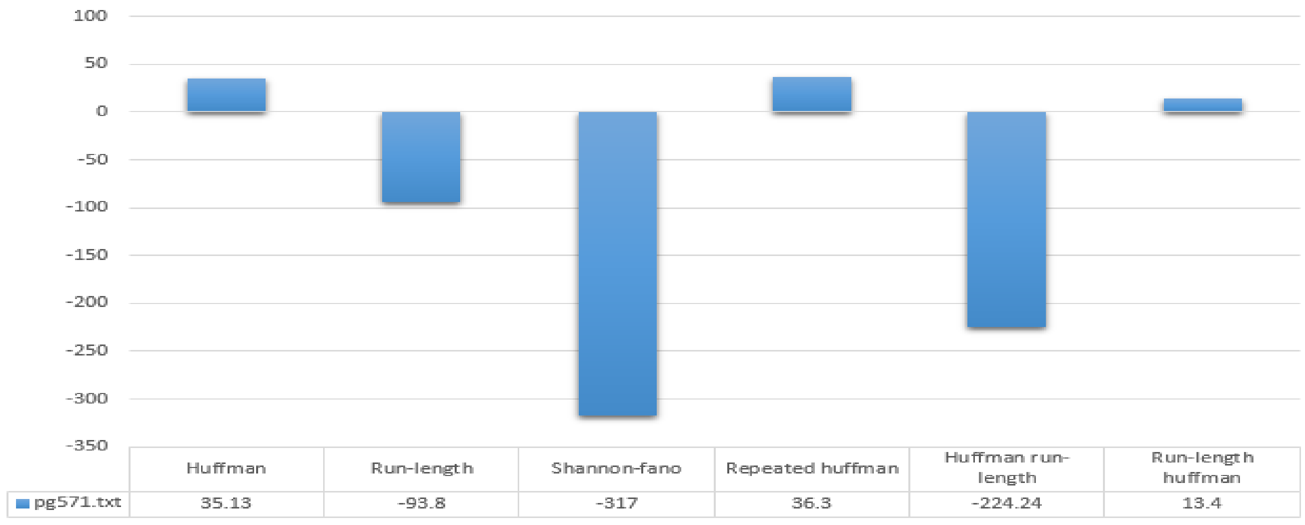
	Huffman	Run-length	Shannon-fano	Repeated huffman	Huffman run-length	Run-length huffman
🟦 boat.jpg	1	-49.7	-465	-13.1	-958	13.9

Fig 4: Results for boat.jpg



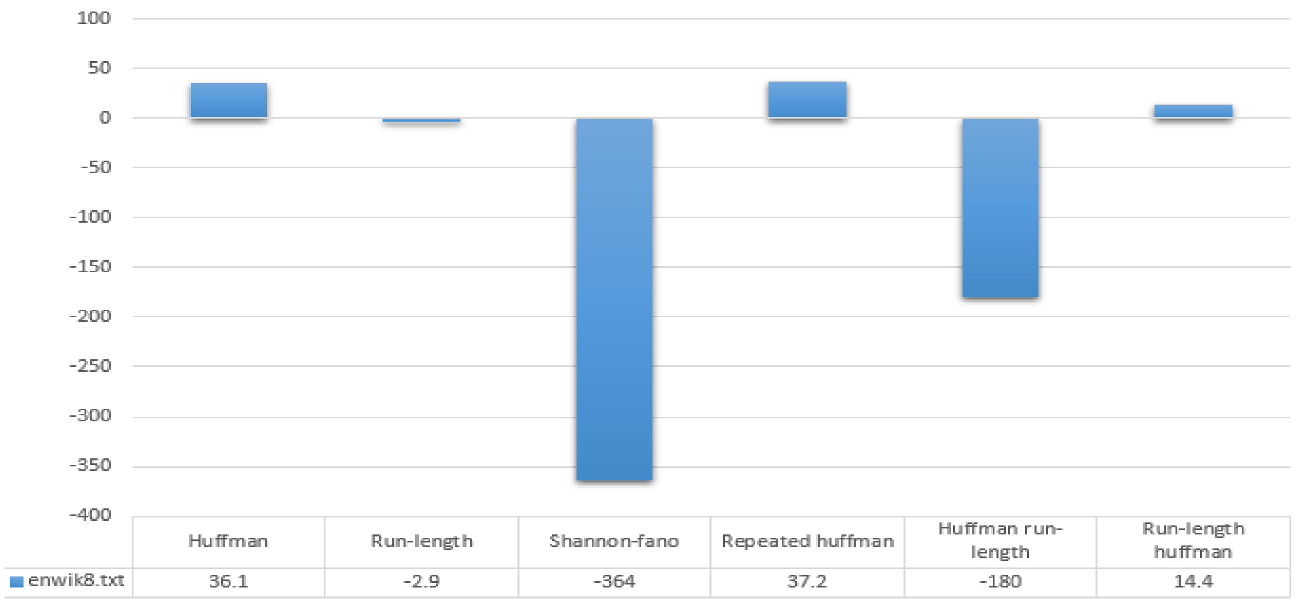
	Huffman	Run-length	Shannon-fano	Repeated huffman	Huffman run-length	Run-length huffman
▬ peppers.jpg	0.38	-54.1	-469	-34.6	-2849	1.9

Fig 5: Results for peppers.jpg



	Huffman	Run-length	Shannon-fano	Repeated huffman	Huffman run-length	Run-length huffman
▬ pg571.txt	35.13	-93.8	-317	36.3	-224.24	13.4

Fig 6: Results for pg571.txt



	Huffman	Run-length	Shannon-fano	Repeated huffman	Huffman run-length	Run-length huffman
▬ enwik8.txt	36.1	-2.9	-364	37.2	-180	14.4

Fig 7: Results for enwik8.txt

Fig 8 describes that all the compression techniques are giving better result for enwik8 text file where total 7 methods have used including recent and proposed compression techniques. All the compression techniques are described in the previous chapters. After computing all the results the assumption comes that all the compression techniques are better for text file than image file.

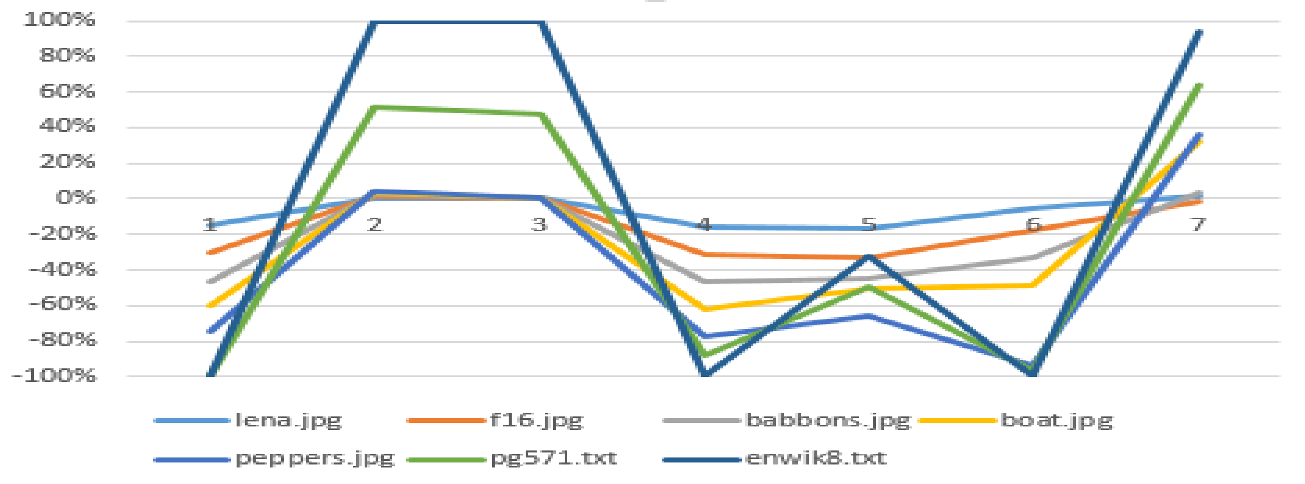


Fig 8: Comparison between benchmark files

6. CONCLUSION

Data compression is a process that reduces the data size, removing the excessive information and redundancy which consequently reduces the storage space, cost & increases the Data transfer rate in communication. In this paper recent methods are combined in a single method. The unique 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 "Repeated Huffman", which gives better result even from Huffman for text data. On the other hand Run-Length Huffman gives a better result in comparison with other proposed method for both image and 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 "Huffman + Run-length" algorithm, which only manages a Compression ratio (%) in negative magnitude. Besides, Shannon fano always gives the worst result among all the methods. At long last, it remains space for the future research and development on several fields which can be carried on like LZW Huffman, Huffman LZW, LZW Run-Length and Run-Length 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.

7. ACKNOWLEDGMENTS

Our thanks to Almighty and my supervisor for his continuous guidance, encouragement, and patience, and for giving me the opportunity to do this work.

Finally, my deepest gratitude and love to my parents for their support, encouragement, and endless love.

8. REFERENCES

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

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

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

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

[5] S. Porwal, Y. Chaudhary, J. Joshi and M. Jain , " Data Compression Methodologies for Lossless Data and Comparison between Algorithms" International Journal of Engineering Science and Innovative Technology (IJESIT) Volume 2, Issue 2, March 2013.

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

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

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

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

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

[11] 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.

[12] 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.

[13] 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.

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

[15] http://www.gutenberg.org/cache/epub/571/pg571.txt.

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

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

[18] 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.

[19] Ostadzadeh, S. A. and Elahi, B. M. and Zeialpour, Z. T, and Moulavi, M. M and Bertels, K. L. M, : A Two Phase Practical Parallel Algorithm for Construction of Huffman Codes, Proceedings of International Conference on Parallel and Distributed Processing Techniques and Applications, pp. 284-291, Las Vegas, USA, June 2007.

[20] 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.

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

[22] Doa'a Saad El-Shora & Ehab Rushdy Mohamed. A "Performance Evalution of Data Compression Techniques Versus Differenct Types of Data" . Article : (IJCSIS) International Journal of Computer Science and Information Security, Vol. 11, No. 12, December 2013

[23] Kashfia Sailunaz, Mohammed Rokibul Alam Kotwal and Dr.Mohammad Nurul Huda. Article: Data Compression Considering Text Files. International Journal of Computer Applications 90(11):27-32, March 2014. Full text available.