

Original Research Article

Plain Text Encoding/Decoding Technique Using a Combination of Huffman and Run-Length Algorithms.

ABSTRACT

This work is devoted to study the effect of applying a hybrid encoding/decoding algorithm to textual data. The sole purpose is to analyze the effect on the size as well as the complexity of the output encoded data. The proposed combination is that of Huffman and Run-Length algorithms. This study focuses on the sequence of applying the two algorithms to see if it has an effect on the output data or not, and the impact of input data format on the result. Results show that the data format and the sequence in which the algorithms are applied actually affect the output. Moreover, it is shown why these two algorithms were chosen and each of them contribute to the overall result.

Keywords: *Encoding techniques, Huffman and Run-Length Algorithms, Compression techniques, encoding, decoding.*

1 INTRODUCTION AND RELATED WORK

In recent days dealing with data no matter of their forms like plain text or binary data is a concrete problem. Data can be stored in local devices or resources or transmitted across networks. So, what if transmission path is unsafe or the storage devices can be hacked? The common answer is that data will be insecure. From this point of view the trouble can be figured. **The major problems are the safety and size of data.** In fact, the problem could not be solved unless having an algorithm or technique that could facilitate the process of maintaining data by scaling down the size and representing the data in some other secure form without losing the original data. So using cryptography will help solving the mentioned problem and accomplishing the main goal. **The domain of this paper is to encode/decode the input data using Huffman and Run-Length techniques.**

Encoding is a field of study which deals with the secret transmission “transforms data into another format” of messages/data between two end-users. It uses schemes that are publicly available so it can be reversed easily and it is for maintaining data usability.

In fact, one of the most significant problems facing the digital data in general is the size of data and its protection. All types of data should be stored, sent, and received in a secure form with little size as much as it can be. Many algorithms are presented to solve this problem.

To be more specific, converting plain text to encrypted or cipher text using a unique encryption key is called encryption [12]. Which means encryption/decryption of data is only possible with the corresponding key. The encryption/decryption issue will not be discussed during this job.

One of the simplest techniques for lossless data compression is Run-length encoding (RLE). It reduces strings by recurring characters to a single character. That means the run of characters “plain text source” is replaced with the number of the same characters and a single character which is the RLE principle. The RLE performance is based on a sequence of identical values of the input data [6-7].

Some other technique for lossless data compression is Huffman encoding. The Huffman algorithm is the one of earliest data-compression and encryption algorithms. It developed by David A. Huffman in 1979. The Huffman algorithm yields a variable and fixed-length binary code depending on the probabilities of each symbol of a source alphabet. The proof [3] that Huffman code can be surprisingly

difficult to cryptanalyze motivated us make a decision of using Huffman encoding as a second main step of changing the form of original plain text and as a private-key “closed key” of encoding/decoding. The final step is to cipher the output data obtained by applying Huffman algorithm using RLE algorithm. As well, the RLE algorithm will be applied to reduce the size obtained data.

Our technique process can be summarized as follows: the original plain text will be encoded as ASCII code in binary form, then represented as a binary code using Huffman encoding. Finally, the RLE will check the repeating string of characters to produce non-repeated data with new cipher form. That will be done because of the properties of Huffman and RLE algorithms that have been discussed.

The work of Rezaul (et. al) [11] shows an efficient decoding technique for Huffman codes and presents a novel data structure for Huffman coding in which in addition to sending symbols in order of their appearance in the Huffman tree one needs to send codes of all circular leaf nodes, the number of which is always bounded above by half the number of symbols

A new devised algorithm to hide text in an image using Huffman encoding and 2D Wavelet transform is presented by Saddaf Rubab (et. Al) [12]. The paper [12] discusses the Huffman Algorithm as a two-part process. The first part is an encoding process. The process starts by set of symbols/letters and their respective frequencies in ascending or descending order. Each symbol/letter with its frequency is a leaf node at the start. Selecting two symbols with smallest frequencies is the next step. The process will continue by adding their frequencies and assign it to parent node, until only one node remains which is called the root node. The first process will finish by assigning binary 0's and 1's to all the nodes. The second part is the decoding process which starts by creating a Huffman table, which is used to decode symbols/letters in original data using the generated codes.

In [13] a new approach of run length encoding (RLE) is proposed to compress discrete cosine transform (DCT) coefficients of time domain ECG signals. Energy compaction property of DCT facilitates the process of length encoding by accumulating the correlative coefficients into separate segments. Thus the high probability of redundancies in consecutive coefficients facilitates the use of RLE. To increase the CR, two stages of RLE are performed on the quantized DCT coefficients. Then a binary equivalent of RLE values are obtained by applying Huffman coding. The conclusion shows that the performance of a compression scheme can vary with the characteristic of the input data set. So, for the same bit rate, the RLE based compression scheme achieves different distortion indices for different databases [13].

The paper [14] proposes a new algorithm for data compression, called j-bit encoding (JBE). An experiment by using 5 types of files with 50 different sizes for each type was conducted, 5 combination algorithms has been tested and compared. The proposed algorithm gives better compression ratio when inserted between move to front transform (MTF) and arithmetic coding (ARI).

Compression Using Huffman encoding article [1] discussed the various techniques available for Lossless compression. The analysis of Huffman algorithm and comparison with various Lossless compression techniques as Arithmetic, LZW and Run Length Encoding has been stated. Comparison of common algorithms with Huffman Encoding has been performed from different points of view as the basis of their use in different applications and their advantages and disadvantages. The main conclusion is that the Huffman algorithm is used in JPEG compression. It produces optimal and compact code, but relatively slow. Huffman algorithm is based on a statistical model that adds to overhead. As well, the researcher concluded that arithmetic encoding is really effective for more frequently occurring sequences of pixels with fewer bits and reduces the file size dramatically. RLE is simple to implement and fast to execute. LZW algorithm is more adept to practice for TIFF, GIF and Textual Files.

Comparison of lossless data compression algorithms for text data article [2] tested six lossless data compression algorithms and compares their performance. A set of selected algorithms such as Huffman Encoding, Shannon Fano Algorithm, RLE, LZW, and Adaptive Huffman Algorithm are examined and followed out to evaluate the performance in compressing text data. Although they are tested for ten text files with different file sizes and different contents. The compression behavior depends on the category of the compression algorithm: lossy or lossless. Compression ratio, factor

and time, and saving percentage are used to evaluate the performance of discussing lossless algorithms. It is important to mention that the performance depends on the type and the structure of the input source file. The main article's conclusion is that the Shannon Fano algorithm can be viewed as the most efficient algorithm comparing to other discussed algorithms. The reset value of this algorithm exhibit in an acceptable scope and it better results for the files with large size. Besides the article depicts the all discussed algorithms work well, except Run length encoding algorithm. Finally, the LZW algorithm does not work well for large file size.

The procedure of converting images into text format can be done. Lossless Huffman Encoding Technique for Image Compression and Reconstruction Using Binary Trees discussed the way of saving a black and white image, which its pixels of different shades of grey. Each pixel has a number value corresponding to the brightness or darkness of the shade. That stands for the pixel colored with black is 0 and white is 255, and all the numbers in between the black and white colors are shades of grey. So, each pixel is coded as some integer from 0 to 255 [9].

A digital image can be coded with respect to a model using Huffman encoding. It's well recognized that the Huffman's algorithm generates minimum redundancy codes compared to other algorithms. The presented scientific article presents a compression and decompression technique based on Huffman encoding and decoding for scan testing to reduce test data volume, test application time. The main conclusion is that the image compression method is well suited for grey scale (black and white) bitmap images [8].

Breaking a Huffman Code article examines the problem of deciphering a file that has been Huffman coded. The authors find that a Huffman code can be surprisingly difficult to cryptanalyze. The article introduces the analysis of the situation for a three-symbol source alphabet in details [3].

As a matter of fact, an image or audio file can be converted to a form of plain text to use as an input source. For instance, it is possible to convert any image into editable text file with software such as JiNa OCR Image to text. Furthermore an intermediate programmer can produce an application that will read each pixel in an image. Then save the information about the pixels that been read in text form.

The major contribution of this paper is developing an algorithm that will encode/decode digital data and reduce the input file size.

The sections of this paper are formed as follows: Part I shows the introduction, problem, goal, and related work. Part II represents suggested technique that will be used in the experiment. Part III discusses the experiment on some case studies. Part IV represents the conclusions and future work.

2 METHODOLOGY

2.1 Technique

Foremost of all, the algorithm can be described as the set of well-defined instructions that allows a computer to execute a specific task in a specific order [5] "well-defined procedure". The technique is a procedure to complete a specific task.

Figure 1 shows the process sequence that manipulates the data presented as a plain text. Figure 2 depicts the steps of the proposed technique.

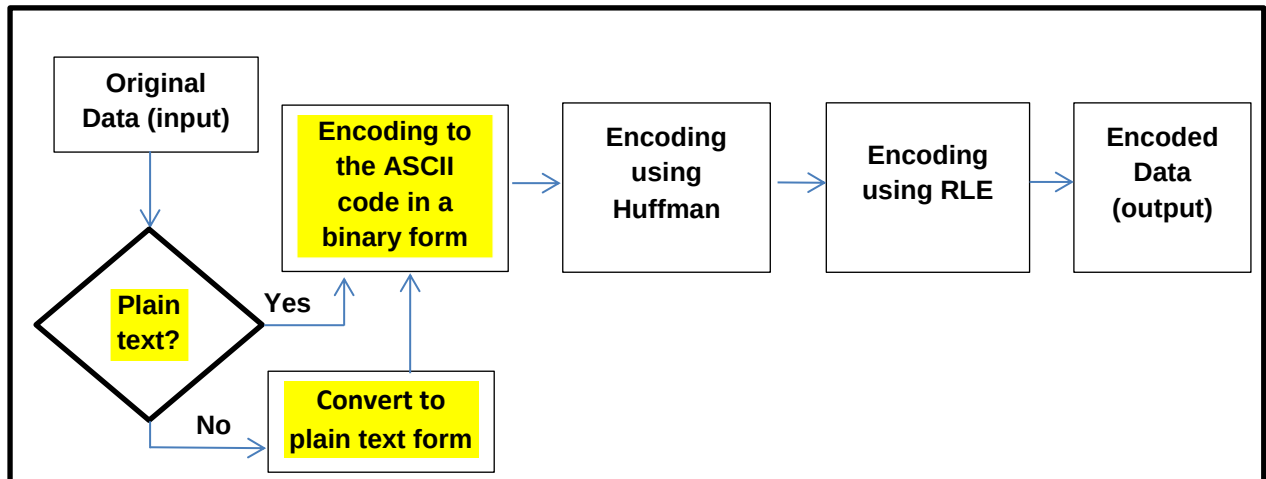


Fig. 1. Process of **Encoding** using Huffman and RLE techniques

Step 1: Start.
 Step 2: Recognize the type of original data "source".
 Step 3: If the type of original data is plain text, go to step 5.
 Step 4: If the type of original data is non-plain text (e.g. Image), then represent the original data in plain text form.
 Step 5: Encode the original plain text to the ASCII code in a binary form.
 Step 6: Represent the original plain text in Huffman form.
 Step 7: Represent the data that's been encoded using the Huffman (Step 6) in RLE form.
 Step 8: Compare the size and form of the original data with the size and form of output data in step 7
 Step 8: Calculate the saving percentage %.
 Step 9: Stop.

Fig. 2. Encoding/Decoding technique - steps

The application that allows to implement the suggested Encoding/Decoding technique was developed using NetBeans IDE. The developed application allows the user to pick a text file and represent it in Huffman and/or RLE form. All the obtained data will be saved in Huffman file and/or RLE text file. The information about the input/output text file as title, size, date, time, and used algorithm will be saved in the database to be used later on for comparison.

2.2 Experiments

This section depicts the comparison table of experiments that were done on different original data. The tables (Table 1 and Table 2) show the original plain text, its size, and output data "in final step". It is quite apparent that the data and its size change throughout the executed process.

The formula used to calculate the shrinkage of the source data as a percentage is [2]:

$$\text{Saving percentage} = \frac{\text{Size before compression} - \text{Size after compression}}{\text{Size before compression}} \% \quad (1)$$

2.3 Results And Discussion

As seen in Table 1 the original data is changed to a different form after implementing Huffman and RLE algorithms respectively. Actually, implementing RLE directly on original data will change the input data in RLE form, but using Huffman coding followed by RLE to encode and find the repeated characters “recurring characters to a single character”, will change the input “original” data to a form that is characterized with more compression factor.

But what will happen if the suggested sequence process is changed, using the same input as in table 1; i.e. implementing RLE then Huffman algorithms on the same original data?

As examined in Table 2, the original data is changed to a different form after implementing RLE and Huffman algorithms respectively. It is obvious, that the size of the output data is greater than that of the output data obtained previously in table 1. From the point of view of encoding/decoding and the output data size, this proves that the suggested sequence process “Huffman-RLE” gives better results. Likewise, table 1 shows that better outcomes will be achieved if the type of the input data is ASCII code in binary form.

In fact, examining the idea of using the Huffman coding before the RLE algorithm could be explained as follows: the Huffman binary code of a character will not always take the same form of a binary code, it depends on its frequency and position in a word. For example, the Huffman binary code of “o” in the word “room” is 1 and the Huffman binary code of “o” in “room and moon” is 11, whereas the Huffman binary code of “o” in “domain+” is 00. Therefore, the outcomes of Huffman coding is not a static value. In addition the Huffman coded text differs from ASCII encoded text. For example the Huffman coded text of “room” is 110010, and the ASCII encoded text is 01110010011011110110111101101101. So, these two mentioned reasons give the suggested sequence process more validity.

As a consequence of what’s been discussed above, the character will have a different binary form using the Huffman algorithm.

Table 1. Encoding and data size changing using Huffman then RLE algorithms respectively

Source data type	Source data view (plain text)	Encoding using Huffman algorithms	Output data view (Encoding) “Implementing Huffman and RLE algorithms respectively”	Encoding/ Decoding (pass or fail)	Original size Bytes	Output size Bytes	Saving Percentage %	Output size (increased/decreased)
The String “Alphabet”	Princess Nourah bint Abdulrahman University	100101100000011 111111010010101 010111000110001 111001111001011 001011101101000 00111110111001 010110101000110 011110110110010 110010100001011 011111011011101 100011111001001 100101000011110 01011	11201110216081101 12011101110111011 10313021304120412 01110212011103110 21101150611031201 11011102110111011 30212041102110212 01110212011101140 11102110511021103 11021305120112021 20111011404120111 021	<u>Pass</u>	43	190	-341	↗

ASCII code in binary form Of "Princess Nourah bint Abdulah man University "	010100000111001 001101001011011 100110001101100 101011100110111 001100100000010 011100110111101 110101011100100 110000101101000 001000000110001 001101001011011 100111010000100 000010000010110 001001100100011 101010110110001 110010011000010 110100001101101 011000010110111 000100000010101 010110111001101 001011101100110 010101110010011 100110110100101 110100011110011	101011111000110 110010110100100 011001110010011 010100011001000 110011011111101 100011001000010 001010100011011 001111010010111 110111111001110 110010110100100 011000101111011 111101111101001 110110011011100 010101001001110 001101100111101 001011110010010 100111101001000 111011111101010 101001000110010 110100010011001 101010001101100 011001001011010 00101110000110	11101110513021102 12011102110112011 30212031201120211 01110113021201130 21202110611021302 12011401130111011 10113021102120411 01120111051106120 31102120111021101 12011302130111041 10611051101120311 02120211031301110 11101120112031302 11021204110112011 10412011201110112 04110112011303110 61101110111011101 12011302120111021 10113011202120211 01110113021102130 21201120111021101 130111031402110	<u>Pass</u>	343	372	-8	↗
--	---	--	--	-------------	-----	-----	----	---

195 ↗ - increase in data size and ↘ - decrease in data size

196

197

198

199

200

Table 2. **Encoding** and data size changing using RLE and Huffman algorithms respectively

Source data type	Source data view (plain text)	Encoding using RLE algorithms	Output data view (Encoding) "Implementing RLE and Huffman algorithms respectively"	Encoding/ Decoding (pass or fail)	Original size Bytes	Output size Bytes	Saving Percentage %	Output size (increased/decreased)
The string "Alphabet"	Princess Nourah bint Abdulrahman University	1P1r1i1n1c1e2s1 1N1o1u1r1a1h1 1b1i1n1t1 1A1b1d1u1l1r1a1h 1m1a1n1 1U1n1i1v1e1r1s1i1 t1y	01111111011101011 11001000011001101 10111111000111111 00101101001000111 11100101000111010 11010010101010110 11100101111001000 01001101011011001 00111001011000101 01000110110001110 10110100101010110 11010110100100001 01101100000100001 11100111000001101 11011101011111001 11100100110100101	<u>Pass</u>	43	272	-532	↗

ASCII code in binary form Of "Princess Nourah bint Abdulrah man University "	010100000111001 001101001011011 100110001101100 101011100110111 001100100000010 011100110111101 110101011100100 110000101101000 001000000110001 001101001011011 100111010000100 000010000010110 001001100100011 101010110110001 110010011000010 110100001101101 011000010110111 000100000010101 010110111001101 001011101100110 010101110010011 100110110100101 11010001111001	101110115031201 120211011201110 211031202130211 021201110111031 202110312021201 160112031202110 411031101110111 031201120214011 102110115011602 130112021101120 111021103120311 011401160115011 102130112021201 130311011101110 211021303120112 021401110211011 402110211011102 140111021103130 116011101110111 011102110312021 101120111031102 120212011101110 312011203120211 021101120111031 101130412011	01000010001100101 01101011110001111 01110010001111000 01011100101101011 11011101101101110 01011101111000010 00010110101111011 10010110101111011 10111100011001110 00111101101011110 11100101100000101 10100100001000010 11010111100011110 11101100010000101 11001000110010100 01100111011101101 10001111011100100 01111000010111001 01101011110110100 10001100010001100 11100011001010000 10111011011000111 10111011110001101 10110100100001000 01011100101110110 11011010111100011 11011101100010000 10111001000110001 01110010111001000 01011101100010000 1011100101101 ...	<u>Pass</u>	344	714	-107	↗
---	--	--	---	-------------	-----	-----	------	---

201 ↗ - increase in data size and ↘ - decrease in data size

202

203 2.3.1 Results

204 The two tables presented above show some results that could be summarized as follows:

205 1. Encoding/Decoding is passed because of changing the original data view as seen in output data
206 view field. Which means the main aim is passed.

207 2. Encoding the input using Huffman encoding before RLE is implemented and makes the process
208 more reliable.

209 3. After accomplishing the process:

210 3.1 If the input data is a plain text, the size of data grows exponentially. In first experiment, the size
211 of the original data which is in text format is 43 bytes and the size of output data is 138 bytes.
212 This means, the size of original data has increased 341%.

213 3.2 If the input data is an ASCII code in binary form, the size of data grows slightly. The second
214 experiment in table 1 shows that the size of the original data which is in binary format was
215 increased 8%.

216 4. The reason behind increasing the size of output data is that the Huffman data has no long
217 sequences of frequented 0's and 1's in table 1 and most of RLE frequencies is 1 in table 2.

218 5. Implementing the RLE before Huffman coding of the original data gives bad results as seen in
219 table 2.

220

221 **3 CONCLUSIONS AND FUTURE WORK**

222 The proposed technique can be utilized to encode and decode data and to increase security level
223 during data communication. The encoded data is surprisingly difficult to cryptanalyze. The outcome is

totally different compared with the original data. The decoding cannot be attained without using the Huffman-RLE algorithm sequence. So, as a result, Encoding/decoding is done and passed if the input is an ASCII code in binary form, and the proposed technique increases the size of data. This technique may be improved using the adaptive Huffman encoding technique or by optimizing the Huffman code, which will happen when the probability of each symbol is a negative power of two [10].

REFERENCES

1. Mamta Sharma. Compression Using Huffman Encoding. IJCSNS International Journal of Computer Science and Network Security. 2010; 10(5): 133-141.
2. S.R. Kodituwakku, U.S. Amarasinghe. Comparison of lossless data compression algorithms for text data. Indian Journal of Computer Science and Engineering, ISSN : 0976-5166, 2001; 1(4) 416-425.
3. David W. Gillman, Mojdeh Mohtashemi, and Ronald L. Rivest. On Breaking a Huffman Code. IEEE - TRANSACTIONS ON INFORMATION THEORY, 1996; 42(3): 972-976.
4. Nigam Sangwan. Text Encryption with Huffman Compression. International Journal of Computer Applications (0975–8887). 2012; 54(6): 29-32.
5. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. Introduction to Algorithms. Third printing, MIT Press. 2002.
6. A. Nagarajan, K. Alagarsamy. An Enhanced Approach in Run Length Encoding Scheme", International Journal of Engineering Trends and Technology. ISSN: 2231-5381. 2011; 2(1): 43-47.
7. S. Sarika, S. Srilali. Improved Run Length Encoding Scheme For Efficient Compression Data Rate. Int. Journal of Engineering Research and Applications. ISSN: 2248-9622. 2013; 3(6): 2017-2020.
8. Pujar, Jagadish H.; Kadlaskar, Lohit M. A new lossless method of image compression and decompression using Huffman encoding technique. Journal of Theoretical and Applied Information Technology. 2010; 15 (1/2): 18-23.
9. Mridul K. Mathur, Seema Loonker, Dheeraj Saxena. Lossless Huffman coding technique for image compression and reconstruction using binary trees. ISSN:2229-6093. 2012; Vol 3 (1): 76-79.
10. Al-hmeary B.A. Role of Run Length Encoding on Increasing Huffman Effect in Text Compression. Journal of Kerbala University. ISSN: 18130410. 2008; Vol. 6 (2): 199-204.
11. Rezaul Alam Chowdhury, M. Kaykobad, and Irwin King. An efficient decoding technique for Huffman codes. Information Processing Letters. Elsevier Science. 2002; 81; 305–308.
12. Saddaf Rubab and Dr. M. Younus. Improved Image Steganography Technique for Colored Images using Huffman Encoding with Symlet Wavelets. IJCSI International Journal of Computer Science Issues. ISSN (Online): 1694-0814. 2012; Vol. 9 (2-1), 194-196.
13. S. Akhter and M. A. Haque. ECG COMPRESSION USING RUN LENGTH ENCODING. IEEE - Signal Processing Conference, 18th European. ISSN: 2219-5491. 2010; 1645 – 1649.
14. I Made Agus Dwi Suarjaya, A New Algorithm for Data Compression Optimization.(IJACSA) International Journal of Advanced Computer Science and Applications. 2012; Vol. 3 (8), 14-17.