Research Journal of Applied Sciences, Engineering and Technology 10(7): 736-741, 2015
DOI:10.19026/rjaset.10.2425
ISSN: 2040-7459; e-ISSN: 2040-7467

Submitted: July 7, 2013 Accepted: August 21, 2013 Published: July 10, 2015

Research Article

Does an Arithmetic Coding Followed by Run-length Coding Enhance the Compression Ratio?

[1]Mohammed A. Otair and [2]Ahmad M. Odat
[1]Faculty of Computer Sciences and Informatics, Amman Arab University, Amman-Jordan
[2]Faculty of Science and Information Technology, Irbid National University, Irbid-Jordan

Abstract: Compression is a technique to minimize the quantity of image without excessively decreasing the quality of the image. Then, the translating of compressed image is much more efficient and rapidly than original image. Arithmetic and Huffman coding are mostly used techniques in the entropy coding. This study tries to prove that RLC may be added after Arithmetic coding as an extra processing step which may therefore be coded efficiently without any further degradation of the image quality. So, the main purpose of this study is to answer the following question "Which entropy coding, arithmetic with RLC or Huffman with RLC, is more suitable from the compression ratio perspective?" Finally, experimental results show that an Arithmetic followed by RLC coding yields better compression performance than Huffman with RLC coding.

Keywords: Arithmetic coding, arithmetic vs. Huffman, entropy coding, image compression, run-length coding

INTRODUCTION

The compression is a process for converting the original information into a compressed form without data loosing (Pu, 2006). In other words, compression data means reducing the data file size with preserving the content of the original file, actually this is the main valuable advantage when handling with a huge file size (Kodituwakku and Amarasinghe, 2007). The Data compression concept is mostly compatible with data management in terms of its ability to provide storage space and bandwidth for data transmission (Goetz and Leonard, 1991). Its technique involves encoding information by using a smaller number of bits instead of its corresponding original file. There are two classifications of data compression algorithm: lossless or Lossy. The latter classification usually used statistical redundancy to represent data in brief without losing information by eliminating unnecessary redundancy. Lossless compression algorithm is applicable because most data has statistical redundancy. It helps in optimal using of resources like transmission bandwidth and storage space. The life cycle for compression algorithms are consists of two phases: compressed and decompressed. Decompressed is an extra process will cost more computational processing (Ahmed *et al*., 2013).

Missing or losing data and information are acceptable in lossy data compression algorithm. High quality files and storage space are direct proportion, some application does not care about the quality, instead of that they are looking for reducing the size of files for various purposes, some details can be discarded to save storage space such as multimedia applications. Many features presented by lossless data compression algorithm for space science applications, such as increase the scientific revenue, and reduce the requirement for the data archive volume. Other meaning, lossless data compression algorithm guarantee to reconstruct and rebuild the original data file without losing or missing any bit. Lossless data compression algorithm preserves the original data file in accurate and complete form, this process done by removing unnecessary redundant data from the source file. Decompression process suppose to retrieve all the deleting redundancy data to rebuild/reconstruct again the original source data, decompression process files requires to obtain the same file as it was before compression, as a result of decompression is to get a replica file for original one (Ahmed *et al*., 2013).

As one of the able-bodied accepted methods of lossless compression is Huffman coding (Huffman, 1952). In this technique, it is supposed that intensity each pixel is associated with a certain probability of appearance and this probability is spatially fixed. Huffman coding specifies a binary code to each intensity value, with beneath codes going to intensities with higher probability (Singh, 2010). If the probabilities can be evaluated a priori, then the table of Huffman codes can be fixed at both the encoder and the decoder. However, in most cases the coding table must be sent to the decoder along with the compressed image

Corresponding Author: Mohammed A. Otair, Faculty of Computer Sciences and Informatics, Amman Arab University, Amman-Jordan

data. Other examples of the lossless compression techniques cover Run Length Coding (RLC) (Goloumb, 1966), arithmetic coding (Witten *et al*., 1987; Ahmed *et al*., 2013) and bit plane coding. These compression techniques also have bounded compression ratios. So, they are used only in susceptible applications (such as medical application) where data loss is rejected, or used in conjunction with other techniques.

The proposed technique in this paper is based on two existing algorithms: Arithmetic Coding (AC) Algorithm and Run-Length Coding (RLC). AC algorithm calculates the probability collective function, and then calculates the function of the accumulative distribution for the original sequence. It can be classified as a lossless compression algorithm. The sequence of symbols is set a single arithmetic codeword which synchronize into [0, 1] subinterval. The amount of symbols in the bulletin increases, the breach acclimated to represent it becomes smaller. The second algorithm is Run-Length coding (RLC), which also can be classified as a lossless data compression algorithm. It is used to minimize the number of repeating characters into input-string, encodes a run of symbols into two bytes (symbol, count).

So, this study will Combining this two method of coding in this way, the image (data) will encoded based on normal Arithmetic coding, additional compression can be achieved using Run-Length coding.

LITERATURE REVIEW

Image Compression is a vital component of the available solutions to create image file sizes to be manageable and transmittable. Important criteria such as portability and performance are used in the chosen of the compression and decompression methods. Data compression algorithms can be divided into two groups:

Entropy coding (lossless encoding): Lossless algorithms remove only redundancy existing in the data. The rebuilt image is identical to the original, i.e., all of the information present in the input image has been preserved by compression. Entropy Encoding can be divided into:

- Content Dependent Coding such as Run-length Coding and Diatomic Coding
- Statistical Encoding such as Huffman Coding and Arithmetic Coding

Source coding (lossy encoding): Higher compression is possible using lossy algorithms which create redundancy (by discarding some information) and then remove it.

Hybrid coding (combine entropy coding with source coding): Examples: MPEG, JPEG, etc.

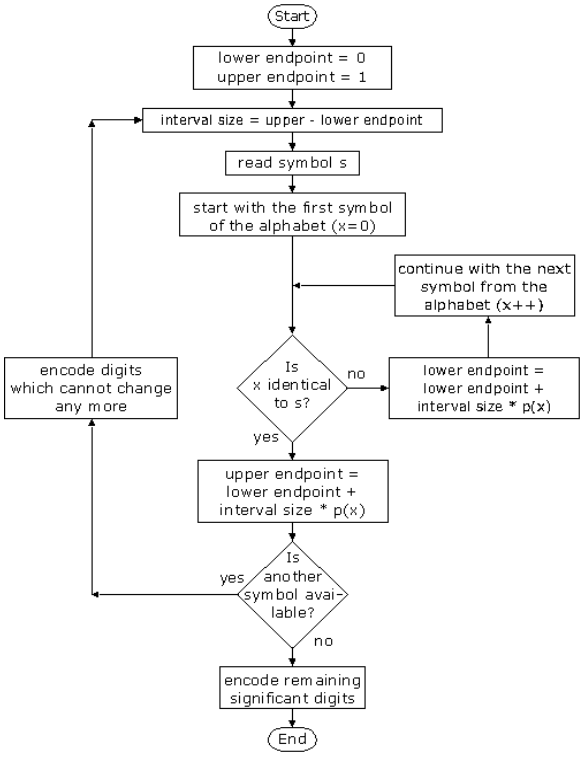

Fig. 1: Arithmetic coding algorithm flow chart

This study focused only on two of the lossless compression techniques: Arithmetic Coding and RLC. The first one is classified as statistical encoding method. The second technique is one of the content dependent coding techniques. So, the proposed method will take the advantages from the two techniques.

Arithmetic coding: Arithmetic coding is variation of coding method called Shannon-Fano-Elias, it is calculate the $p(x^n)$ which is a probability mass function and $F(x^n)$ which is the cumulative distribution function, for x^n as the source sequence in a sub-interval [0,1].

The amount of symbols in the bulletin increases, the breach acclimated to represent it becomes smaller. Sequences of antecedent symbols are encoded together. There is no one-to-one accord amid antecedent symbols and cipher words. Slower than Huffman coding but about achieves bigger compression. An arrangement of antecedent symbols is assigned an individual addition cipher chat which corresponds to a sub-interval in [0, 1]. Figure 1 Asadollah *et al*. (2011) shows the flow chart which describes how the algorithm works.

RLC coding: The second algorithm is Run-Length coding (RLC); also it is a lossless data compression algorithm, it is used to minimize the number of repeating characters into an input-string. Symbols can be encoded by two bytes: one for the symbol, and the second one for the count. The RLC coding method can compress any kind of data but cannot accomplish

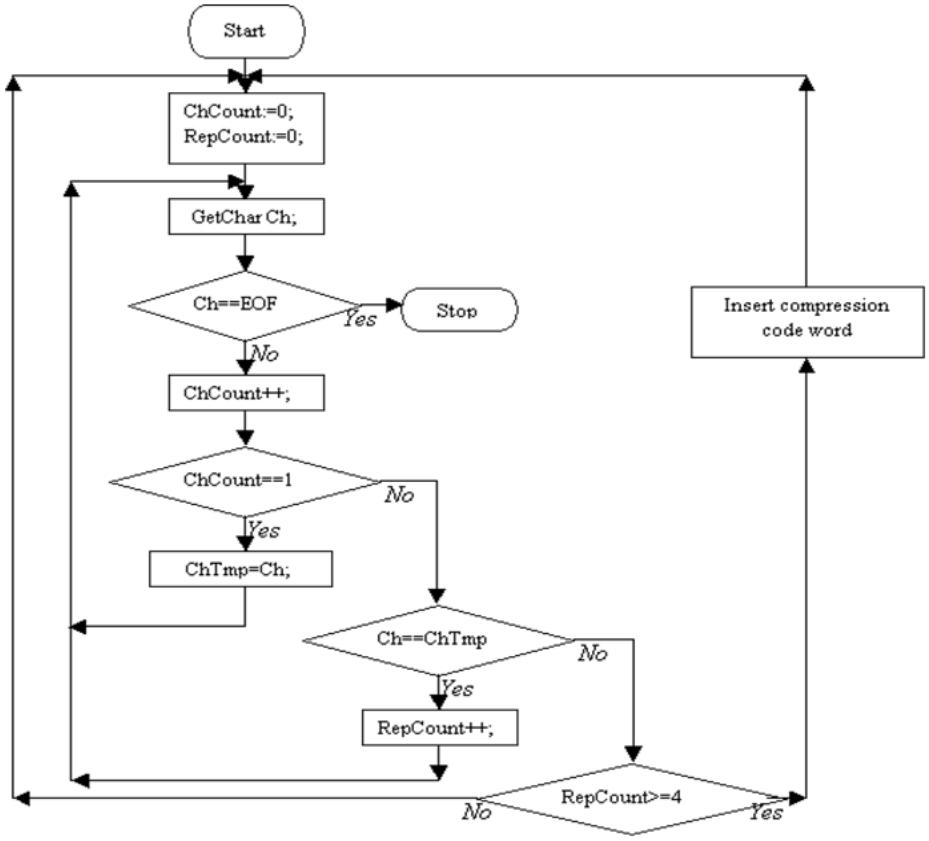

Fig. 2: RLC flow chart

Table 1: Input/output string example using Huffman and arithmetic

Symbol	Probability	Huffman codeword	Arithmetic (sub-interval)
K	0.05	10110	(0.00, 0.10)
A	0.2	00	(0.10, 0.25)
S	0.1	1010	(0.25, 0.37)
P	0.05	10111	(0.37, 0.55)
E	0.3	11	(0.55, 0.85)
R	0.2	01	(0.85, 0.95)
!	0.1	100	(0.95, 1.00)

significant compression ratios in compare with the other techniques. The following Fig. 2 (cpsc.ualr.edu/milanova) shows how it works.

Why used arithmetic coding over Huffman coding? A preferable compression results comes from arithmetic coding because it encodes a message as complete segment rather than separate symbols. The complexity of arithmetic algorithm is $O(n^2)$ but the complexity of Huffman algorithm is $O(n \log_2 n + n \log_2 \log_2 n)$, where n is the number of symbols that been used. So, the complexity of arithmetic algorithm is bigger than Huffman complexity but for this reasons that selected Arithmetic over Huffman: arithmetic accommodates adaptive models and isolate between coding and model. Moreover, it does not need to convert each symbol into a complete number of bits. Nevertheless, it needs a lot of data computation such as division and multiplication. Also, Arithmetic always gives a lowest number of bits in compression.

So, to explain the efficiency of the compression ration using Arithmetic and Huffman; the following example will be used. Table 1 shows the data that been compressed by the two methods (Huffman and Arithmetic).

In result of compression for the same data arithmetic coding give us 17 bit (00011111001001111) code-word, on the other hand Huffman compression for the same data gives 24 bit (00, 00, 1010, 10111, 1010, 1010, 100) code-word.

Abdmouleh *et al*. (2012) introduced a lossless image compression algorithm by merging between Arithmetic coding with RLC.

They employ both the advantages of the Arithmetic coding models (even Adaptive or Static) and the efficiency of the RLC to supply an algorithm which could be helpful in some applications like medical applications. Figure 3 (Abdmouleh *et al*., 2012) shows the framework of their method:

Abdmouleh's method (Abdmouleh *et al*., 2012) is based on the concept that an image is classified or clustered by its similar portions. With RLC, the bit planes that have high weights are classified by sequences of 0 and 1 are sequentially encoded, whereas the other bit planes will be encoded by the arithmetic

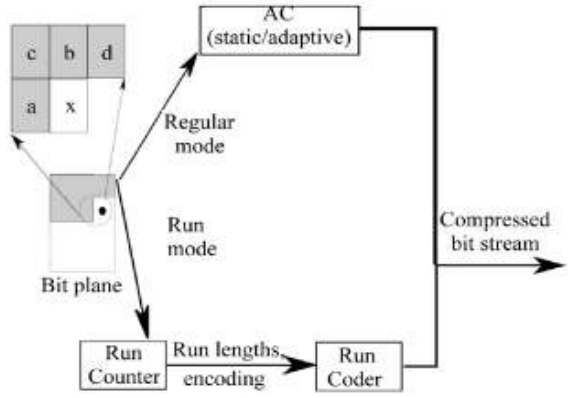

Fig. 3: Merging arithmetic (static/adaptive) with RLE to compress bit plane

coding models. They concluded that by combining Arithmetic coding models with the RLC, a high degree of compression and adaptation efficiency can be achieved.

PROPOSED METHOD

Some compression technique disadvantages: Huffman coding consistently steps rounding errors, because its cipher breadth is belted to several bits. It is also provide to convert every symbol into a complete integral number of bits. However, it suffers from several shortcomings.

Arithmetic and RLC compression technique advantages: Arithmetic Coding does not use discrete number for each bit, instead of that arithmetic coding handling the whole symbols as a one unit. At the other side, in Huffman coding low anticipation symbols use abounding bit, top anticipation symbols use beneath bits. Arithmetic coding is slower than Huffman coding but about achieves bigger compression.

Combining arithmetic and RLC compression technique advantages: This technique of combining two of much effected and fast techniques will achieve a high degree of compression and adaptation efficiency. It will reduce the compression ratio by reducing the number of bit; after apply arithmetic coding on the original data then having the result of the compression process, then after that apply RLC on the compressed data. The following Fig. 4 shows how the proposed method works.

In order to explain how the proposed technique does works, Arithmetic coding and Huffman coding will be applied on Input String: *AASPSS*!. Figure 5 is a graphical representation for Table 2 which describes an arithmetic coding and the distribution its corresponding probability.

According to the Table 2, AC works as follow: every symbol is assigned with its probability within the

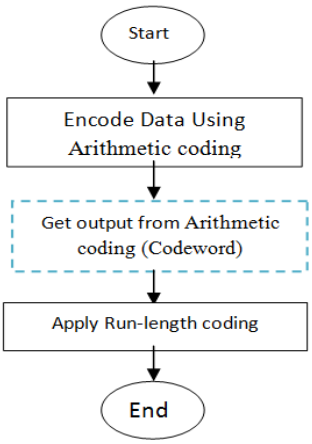

Fig. 4: Proposed method flow diagram

Table 2: Input string example for arithmetic coding

Symbol	Probability	Sub-interval
K	0.1	(0.00, 0.10)
A	0.15	(0.10, 0.25)
S	0.12	(0.25, 0.37)
P	0.18	(0.37, 0.55)
E	0.3	(0.55, 0.85)
R	0.1	(0.85, 0.95)
!	0.05	(0.95, 1.00)

Table 3: Input string example for Huffman coding

Symbol	Probability	Huffman codeword
K	0.1	10110
A	0.15	00
S	0.12	1010
P	0.18	10111
E	0.3	11
R	0.1	01
!	0.05	100

complete corresponding interval. Symbol “A” is taken as the first symbol from the input-string "AASPSS!" and its sub-interval [0.10, 0.25] is chosen. The next symbol “A” is treated as the previous symbol, because they have the same sub-intervals [0.10, 0.25]. The sub-interval [0.115, 0.137] with be the corresponding interval for the symbol "S" based on the AC steps in Fig. 1. The same steps are repeated until the symbol "!" is reached as the last symbol in the input-string. Fig. 5 shows how the sub-intervals and bit stream will be manipulated for each symbol. The last symbol probability will be chosen from the last interval [0.1217] and transformed into its corresponding binary code-word (00011111001001111).

If Huffman Coding is applied at the same above example using the input string and their probabilities as in table 3, then the codeword can be compared that will result from the Huffman with codeword that already resulted from an Arithmetic coding.

Huffman Coding Codeword: 00, 00, 1010, 10111, 1010, 1010, 100; Result: So in arithmetic coding give us 17 bit codeword, but in Huffman 18 bit codeword.

Appling RLC after get the compression code from Arithmetic Coding:

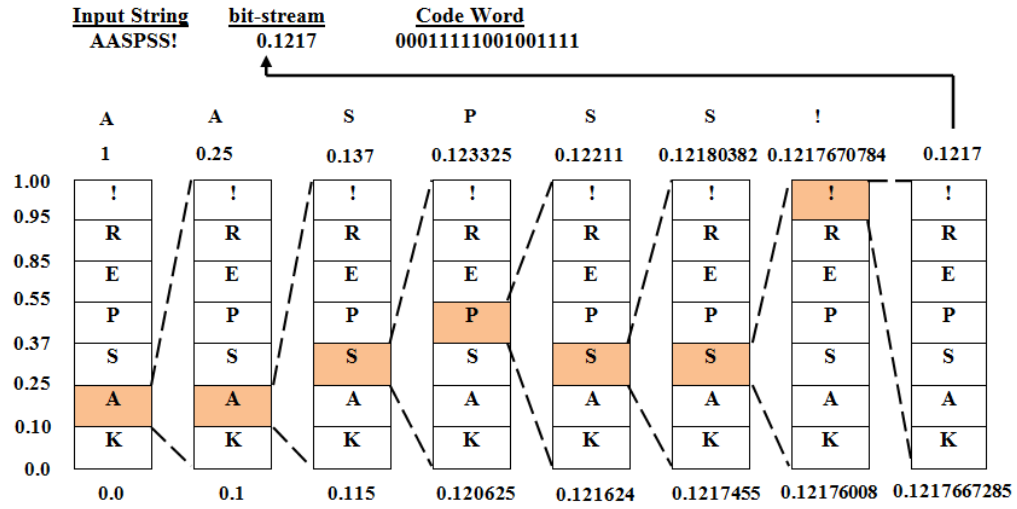

Fig. 5: Arithmetic coding example

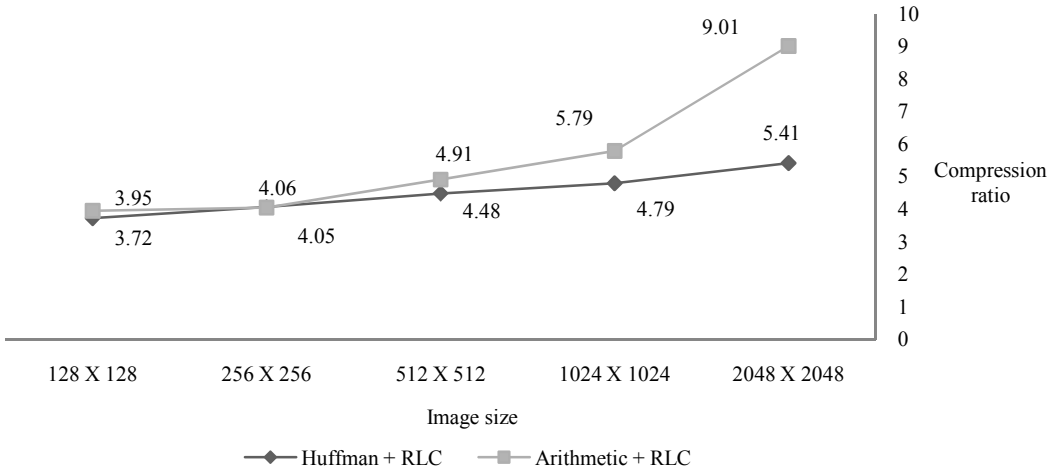

Fig. 6: Comparison of compression ratio for Huffman and arithmetic algorithms using different image sizes

00011111001001111 = (0, 3) (1, 5) (0, 2) (1, 1) (0, 2) (1, 4)

The length of the codeword that resulted from Arithmetic Coding is 17 bit then after applying the RLC the length of the codeword was 12 bit and compression code. However, if RLC was applied after the Huffman coding, the result will be:

000010101011110101010100 = (0, 4) (1, 1) (0, 1) (1, 1) (0, 1) (1, 1) (0, 1) (1, 4) (0, 1) (1, 1) (0, 1) (1, 1) (0, 1) (1, 1) (0, 1) (1, 1) (0, 2)

Then as noted that Huffman followed by RLC either not improve the compression rate or improve it by slight percentage in compare with an Arithmetic followed by RLC. In other words, the performance of AC is mainly similar or better than Huffman technique. Finally implementation of RLC after an Arithmetic coding gives better result than implementation of RLC after Huffman coding for one important thing; that Arithmetic coding always give better compression ratio than Huffman.

EXPERIMENTS ANALYSES

This section explains the experiments achieved in this study which accomplished on test image set consists from different 5 image sizes and 5 samples for each size with bit-depth equals to 8-bits for all of them. The tested images are squared and their sizes are ranging from 2048 to 128. The objective of testing varying sizes of images is to validate the proposed technique. These images are mainly used in image compression and processing fields, because they have a good level of complexity. Matlab programming tool is used to do the experiments which implemented on both algorithms (Arithmetic and Huffman followed by RLC).

Table 4: Average of compression results on test image set

	Compression ratio (bits/sample)	
Test image size	Huffman + RLC	Arithmetic + RLC
2048×2048	5.41	9.01
1024×1024	4.79	5.79
512×512	4.48	4.91
256×256	4.06	4.05
128×128	3.72	3.95

The compression ratio results of all experiments of the implemented algorithms, Huffman and arithmetic coding followed by RLC are summarized in Table 4. The numbers in the second and third columns are the computed averages of the compression ratio of (Huffman+ RLC) and (Arithmetic coding + RLC) respectively. The results show that the average of compression ratios achieved by Arithmetic coding followed by RLC (with different image sizes) are always better than the Huffman coding + RLC.

As shown in the table, Arithmetic Coding followed RLC achieves higher average of compression ratio than Huffman with RLC. The average of compression ratio ranges from 3.95 to 9.01 depending on the image size. It is well known that the compression ratio can be achieved by dividing the original image size by the compressed image.

According to the results in Table 4, a general behaviour can be noticeable, where the increasing in image sizes from 128×128 to 2048×2048, causes an increased improvement of compression ratio averages of the Arithmetic coding more than the Huffman coding. For example, the compression ratio averages of Huffman algorithm for image sizes of 1024×1024 and 2048×2048 was 4.79 and 5.41, respectively. While in Arithmetic coding was 5.79 and 9.01, respectively. Figure 6 is a graphical representation for Table 4 which shows the improvements achieved by the proposed technique.

CONCLUSION

Image Compression will always need new techniques to be implemented, because of the instant need of compression ratio and keeping a quality, not lost any information, and reduce the size with this measure (Quality, Size). So the efficiency of the compression will be increased by combining Arithmetic code with RLC. However, many researchers in the literature concluded that an Arithmetic coding is a time-consumed technique in compare with Huffman. However, from the compression performance perspective arithmetic code gives similar or better efficiency than Huffman technique. As an answer of the main question of this paper we can say that an Arithmetic algorithm followed by RLC improves the compression ratio generally and results better compression ratio than Huffman algorithm.

REFERENCES

Abdmouleh, M.K., M. Atef and M.S. Bouhlel, 2012. A new method which combines arithmetic coding with RLE for lossless image compression. J. Softw. Eng. Appl., 5: 41-44.

Ahmed, S., G. Mehdi and R. Ali, 2013. Large dataset compression approach using intelligent technique. J. Adv. Comput. Sci. Technol. Res., 3(1).

Asadollah, S., B. Ramin, R. Mobin and M. Mostafa, 2011. Evaluation of Huffman and arithmetic algorithms for multimedia compression standards. Int. J. Comput. Sci. Eng. Appl., 1(4).

Goetz, G. and D. Leonard, 1991. Data compression and database performance. Oregon Advanced Computing Institute (OACIS) and NSF Awards IRI-8805200, IRI-8912618 and IRI-9006348.

Goloumb, G., 1966. Run length encoding. IEEE T. Inform. Theory, 12: 399-401.

Huffman, D., 1952. A method for construction of minimum-redundancy codes. P. IRE, 40(9): 1098-1101.

Kodituwakku, S. and U. Amarasinghe, 2007. Comparison of lossless data compression algorithms for text data. Indian J. Comput. Sci. Eng., 1(4): 416-425.

Pu, I.M., 2006. Fundamental Data Compression. Elsevier, Britain.

Singh, V., 2010. Recent patents on image compression: A survey. Recent Pat. Signal Process., 2(2): 47-62.

Witten, I., R. Neal and J. Cleary, 1987. Arithmetic coding for data compression. Commun. ACM, 30: 520-540.