Fast Data Compression Using Huffman-Based Tree Structure

Dilini Lakmali[1],Uththara Waidyarathna[2],Dilanka De Silva[3],Sayansi Sivapatham[4]
Department of Electronics and Telecommunication Engineering
Sri Lanka Technological Campus, Sri Lanka
[1]dilinil@sltc.edu.lk,[2]dhyaniw@sltc.edu.lk,[3]dilankad@sltc.ac.lk ,[4]sayansis@sltc.ac.lk

***Abstract*—There is a high demand to improve the encoding and decoding speed of data compression among modern users. When a binary Huffman code is used, it requires more time to compress or decompress every single bit. The technique which is being proposed in this paper is a new Huffman-Based tree structure instead of a consistent binary tree to reduce decoding time complexity. Since the traversing time depends on the height of the tree, the proposed tree structure provides a smaller height than the height of the regular binary tree. The performance of the regular Huffman tree technique and proposed technique is evaluated in terms of the decoding time. According to the results analyzed, the proposed technique outperforms the current Binary tree technique in terms of decompression speed while the compression performance remains nearly the same.**

***Index Terms*—Compression, Decompression, Huffman tree, Image Processing, Python, Raspberry pi**

I. Introduction

Today's high-performance computing (HPC) applications generate massive amounts of data that are challenging to store and transfer efficiently during execution. As a result, data compression has emerged as a critical technique to mitigate the storage burden, data movement cost. Data compression is divided into two parts as lossless and lossy compression. In lossy compression, some information of the original image is removed. The lossy compression techniques are used in the applications like gaming, entertainment, graphic designing, etc., where the loss in the image/video is acceptable. In lossless compression techniques, the information which cannot be perceived by HVS (Human Visual System) is removed which makes the reconstructed image lossless. However, applications in the fields such as medical imaging, satellite imaging, tissue engineering require preserving the original information of the images where the lossless compression techniques play their role and lossless technique is more preferred than the lossy technique to obtain the original quality of an image. Therefore, to have a better quality performance the Huffman-based technique [1] is introduced. Lossless Image compression is an important research area not only for space requirements but also for reducing query time. Compression and decompression speeds are important aspects of data compression techniques that lead to greater performance. It reduces the performance when it requires more time to compress an image. Even so, It has been carried out more researches based on memory-efficient techniques instead of processing speed. As a consequence, a fast, lossless, and efficient method is needed.

Huffman coding is arguably the most efficient Entropy coding algorithm in information theory, such that it could be found as a fundamental step in many modern compression algorithms. It is the most popular and widely used coding scheme, which collects probabilities of each intensity value in descending order. It assigns shorter codewords [2] for high frequencies and longer codewords for lower frequencies. The Huffman technique assigned a unique codeword for each intensity value and independent of data type.

The traditional Huffman-Based tree structure algorithm utilizes binary code which slows the decoding process [3]. The binary tree [4] structure is currently performed bit by bit decoding. A single bit is compared to any possible code with a codeword length of one. If no match is found, another bit is shifted in to search for the bit pair among all codewords with a word length of two. This process is repeated until a match is identified. Even though, the binary approach is memory efficient, it requires more time to process when the codeword to be decoded is long. In consequence, the research proposes a new Huffman-based tree structure algorithm to reduce the decoding time complexity. The algorithm is based on the variation of the existing Huffman tree structure which encoded each value into a Quaternary code stream instead of a binary bit stream using the Quaternary [3] Huffman tree. A Quaternary code stream for Huffman coding required a shorter Huffman tree which has less depth than the binary Huffman tree. A shorter Huffman tree gives the potential benefit of less traverse time [5], which increases both compression and decompression throughput. Hence, the research carried out compression and decompression algorithm based on the new tree structure. The data sets used in this study are raw images acquired by the Raspberry Pi camera module.

Figure 1 illustrates the process of image (RGB, Bitmap) compression that is captured by the Raspberry Pi camera module. The captured images have raw data with more storage space. Hence, it is required to compress images by considering the space requirement using the Huffman compression algorithm. The RGB images are separated into three layers and apply Huffman coding for each layer individually. Compressed bit stream is decoded using the Huffman decoding algorithm. Finally, the reconstructed image is generated by summing the

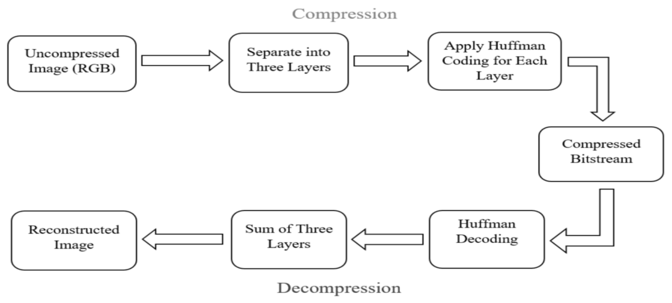


Fig. 1. Systematic Block Diagram of Image Compression

three decoding layers.

The rest of the paper is organized as follows. The literature review has been carried out in section II. Section III describes methodology and materials. Implementation results are present in section IV. Finally, section V concludes the paper.

II. Literature Review

The lossless compression technique ensures the reproduction of original data without any loss of information. In 1952, DAVID A. HUFFMAN presented a coding technique in [6] for data compression, where the codeword is assigned to each symbol and no two symbols have the same codeword. Without requiring any additional information, the starting and the ending point of a symbol can be identified. After the Huffman algorithm was proposed, it was found to be effective in compressing image and video data in addition to text data [7].

According to the Huffman-based studies, some algorithms obtained higher compression ratios by sacrificing processing speed, whereas others achieved higher speeds by sacrificing memory requirement. Hashemian has carried out the compression technique based on the tree clustering algorithm to speed up the process of search for a symbol in a Huffman tree and to reduce the memory size by avoiding the sparsity of the tree. The experiment was conducted on video data and clarify that the method is very efficient [8]. Suri and Goel introduced the use of a ternary tree that is a new one-pass algorithm for the decompression of Huffman codes [9].

Lin, Huang, and Yang have proposed an algorithm by transforming the traditional Huffman tree into a recursion Huffman tree. The algorithm decodes more than one symbol at a time using the recursive Huffman tree and speeding up the decoding time. The decoding time of the proposed technique gives greater improvement instead of using the basic Huffman tree. As the limitation of the study, a large memory is required and only applicable for test data compression problems [10].

In another research, Xiaofeng and Shen have introduced the fast lossless compression scheme for medical images that is based on the LS prediction method and most- likely magnitude Huffman coding with significant time and compression improvements over the JPEG (Huffman) and JPEG2000 (lossless). This newly suggested scheme has the potential to lower the cost of the Huffman coding table while ensuring a high compression ratio [11]. The method enables systems to compress imagery in real-time with software-only implementation.

Jose Oliver and Manuel P. Malumbres introduced a very fast version of the lower-tree wavelet encoder to reduce the processing time that is based on Lower Tree Wavelet Coding using Huffman codes. This includes three stages. At the first stage, all of the symbols required to efficiently represent the transformed image are calculated. Statistics can be gathered during this stage to compute the Huffman table in the next stage. Finally, using Huffman coding the symbols computed during the first stage are coded. The proposed encoder is 9 times faster than progress while the PSNR is from 0.3 to 0.5 dB higher at low bit rates. The encoder was a good fit for real-time multimedia communications with simple hardware and software implementation [1].

In recent research, Rajiv Ranjan introduced the benefits of a DWT-based approach by utilizing the canonical Huffman coding as an entropy encoder. The proposed method requires less computing time and has a compact code-book size compared to the basic Huffman coding [12].

Aharon Fruchtman, Yoav Gross, Shmuel T. Klein, and Dana Shapira has been presented a new variant of Huffman encoding that provably always performs better than static Huffman coding by at least m-1 bits, where m denotes the size of the alphabet. They introduced a new generic coding method, extending the known static and dynamic variants. It is probably as good as the best dynamic variant known to date. The method is shown improvements over static and dynamic Huffman and arithmetic coding even when the encoded file includes the model description [14].

Janarbek Matai, Joo-Young Kim, and Ryan Kastner have proposed a method for energy-efficient Huffman coding which is based on Canonical Huffman coding. Canonical Huffman coding has two major advantages over Traditional Huffman coding. The encoder sends the whole Huffman tree structure to the decoder in basic Huffman coding. As a result, in order to decode each encoded symbol, the decoder must traverse the tree. In Canonical Huffman coding, the decoder only receives the number of bits for each symbol and reconstructs the code word for each symbol. This makes a more efficient decoder in terms of memory usage and computation requirements [15].

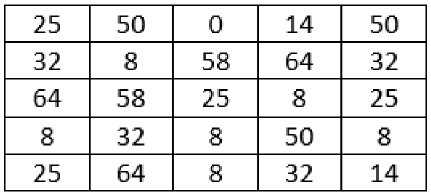
In the above literature, various compression algorithms have been proposed using different Huffman-based coding systems. However, the decoding speed is mostly affected by the length of the Huffman code.

Therefore, this paper introduces a new Huffman tree structure which has not been studied thoroughly in the existing works to produce more efficient optimal code to ensure the time efficiency of Huffman decoding. Due to the less-height tree, it generates a more optimal codeword.

III. Methods and Material

A. Huffman Algorithm

The process begins constructing a Huffman tree [13] from bottom to top. Frequencies of intensity values are assigned to each node of the tree. Then the Binary Huffman tree is built by combining the least frequent nodes. The process is repeated until the last frequency node. After that bit “0” assigns to the left child and bit “1” represents to right child. By traversing the tree, assigns shorter codewords to the higher frequency nodes and longer codewords for lower frequency nodes. The technique assigns a unique codeword for each intensity value. Figure 2 illustrates an example of pixel values from a 5*5 image block. The produced codewords for the example image block using the Binary Huffman tree are shown in Table 1. The constructed Binary Huffman tree is shown in Figure 3.



25	50	0	14	50
32	8	58	64	32
64	58	25	8	25
8	32	8	50	8
25	64	8	32	14

Fig. 2. An example of pixel values of 5*5 image block

TABLE I
CODE WORD GENERATION USING BINARY HUFFMAN TREE

Intensity Value	Frequency	Code-word
8	6	00
25	4	011
32	4	010
50	3	100
64	3	101
14	2	110
58	2	1110
0	1	1111

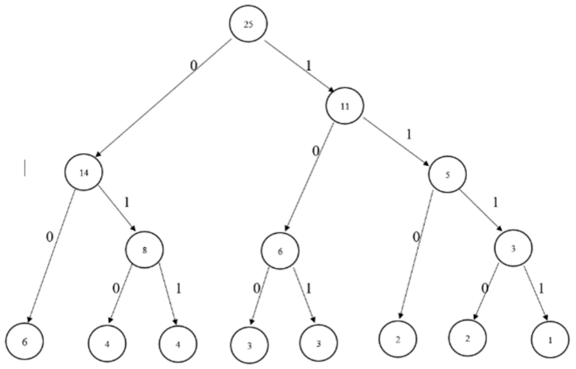


Fig. 3. Binary Huffman Tree

B. Proposed Tree Structure

The research has been proposed a Huffman-Based tree structure to mitigate the processing time. Instead of implementing the Binary Huffman tree structure, the 4-array tree structure is employed. The proposed tree structure is used to produce optimal codewords which speed up the searching process.

The tree has 0 – 4 children nodes named as a left child, left mid child, right mid child, right child. The codewords are made up of the numbers 00,01,10 and 11. To begin, the probabilities are evaluated to all possible pixel values, and select the four-pixel values with the lowest probability. All four-pixel value probabilities are replaced by a single probability node and the probability of the parent node is the sum of these four probabilities. The process is repeated until only one node remains. As the traversing time of a tree depends on its weighted path length, the weighted path length should be minimum to have a minimum time. In addition, the traversing time depends on the height of the tree and symbol frequencies. The height of the Huffman tree is reduced by utilizing the implemented tree structure, compared to the binary tree. Hence the traversing time is mitigated for the petite tree. In this method, the most frequent pixel value is stored first in the header. It has a faster decoding speed since the whole codeword does not need to be stored in the header. Since the decoding process retrieves only two bits at a time, the decoding process is sped up.

After determining the frequencies, the frequencies are stored in a dictionary and used to create the petite tree. Then the pixel values are replaced by the codes. Decoding is achieved by reading decoding data two bits at a time. The novelty technique achieves a faster decoding time for data compression. The process is done to three layers of RGB images separately and combined at the decoding process.

If we consider the set of intensity values as I = I1 ,I2 ,………, In-1 with frequencies F = f0 , f1 ,…. , fn-1 for f0 >f1 >f2 >……>fn-1, Where the intensity value Ii has frequency fi. Using the Huffman algorithm to construct the Huffman tree, the codeword ci, 0 $\leq$ i $\leq$ n-1, for intensity value Ii then can be determined by traversing the path from the root to the leaf node associated with the intensity value Ii, where the left branch is corresponding to “00”, left mid branch is corresponding to “01” and the right mid branch is related to “10” and the right branch is related to “11” [9].

Figure 4 presents the implemented tree for the proposed method which consists of four levels. At the decoding process, it matches two bits at a time from the encoded bit stream by initializing level 1 of the header tree. If there is an intensity value that has a codeword length of 2 bits, it can be found in level 1. However, for the codeword length of 4 bits needs to only match with level 1 and level 2 to find the intensity value. Hence, the simple header tree speeds up the decompression. In table 2, the generated codewords using a 4-array tree structure are shown.

TABLE II
CODE-WORD GENERATION USING PROPOSED HUFFMAN TREE

Intensity Value	Frequency	Code-word
8	6	00
25	4	01
32	4	10
50	3	1100
64	3	1101
14	2	1110
58	2	111100
0	1	111101

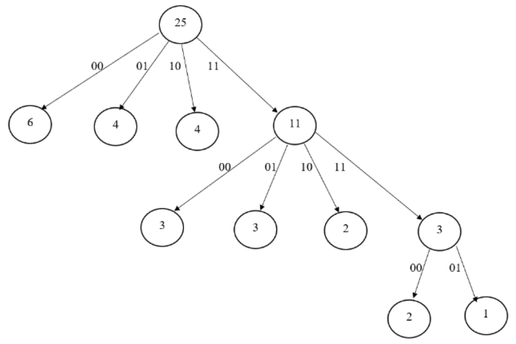


Fig. 4. Proposed Tree Structure

IV. Results and Discussion

In this section, the performance of both techniques are analyzed. The five images with different resolutions are captured through the raspberry pi camera module (5MP, picture resolution 2592 x 1944, Rev 1.3) and compressed using both techniques as the traditional Huffman-based algorithm and the proposed algorithm in the python environment.

The Compression ratio, Compression and Decompression speeds, PSNR were measured using the implemented techniques and the decompression time was mainly analyzed for all images. The compression and decompression speeds of both techniques are measured with the same environment and same compiler. The Raspberry Pi3B (Raspbian OS) and Intel®Core™i5-7200 CPU running at 2.5 GHz with Turbo Boost up to 3.1 GHz (Windows OS) were used. The average output for 10-20 runs was taken in all cases. The 20 consecutive runs for each image in the Windows platform and 10 consecutive runs in the Raspberry Pi platform were considered for the experiment. The analyzed results are shown the below tables.

TABLE III
Performance Analysis of Binary tree technique using Windows platform

		snapshot				
Image Name		1.bmp	2.bmp	3.bmp	4.bmp	5.bmp
Image Size (MB)		14.4	3.6	0.9	0.225	0.025
Compression Ratio		1.0721	1.0579	1.1146	1.1191	1.1113
Compression Time (s)	Min	70.40	16.61	3.87	0.94	0.109
	Max	100.84	19.35	4.7	1.24	0.22
	Avg	77.65	17.85	4.22	1.048	0.138
Decompression Time (s)	Min	170.79	41.72	10.47	2.65	0.28
	Max	211.074	47.56	11.98	3.31	0.48
	Avg	184.51	44.80	11.22	2.95	0.338
PSNR (dB)		30.77	29.28	31.25	31.92	30.31

TABLE IV
Performance Analysis of Proposed Tree Technique using Windows Platform

		snapshot				
Image Name		1.bmp	2.bmp	3.bmp	4.bmp	5.bmp
Image Size (MB)		14.4	3.6	0.9	0.225	0.025
Compression Ratio		1.0540	1.0414	1.0945	1.0979	1.0903
Compression Time (s)	Min	70.88	16.85	3.89	0.94	0.109
	Max	82.81	26.69	5.39	1.21	0.172
	Avg	75.26	18.89	4.37	1.05	0.12
Decompression Time (s)	Min	63.13	15.86	3.77	0.95	0.109
	Max	69.93	18.58	4.75	1.75	0.265
	Avg	66.75	16.65	4.11	1.101	0.137
PSNR (dB)		30.77	29.28	31.25	31.92	30.31

TABLE V
Performance Analysis of Binary Tree Technique using Raspberry Pi

		snapshot				
Image Name		1.bmp	2.bmp	3.bmp	4.bmp	5.bmp
Image Size (MB)		14.4	3.6	0.9	0.225	0.025
Compression Ratio		-	-	1.1146	1.1191	1.1112
Compression Time (s)	Min	-	-	26	6.65	0.92
	Max	-	-	26.71	6.84	1
	Avg	-	-	26.31	6.75	0.947
Decompression Time (s)	Min	-	-	59.45	14.91	1.58
	Max	-	-	61.89	15.32	1.69
	Avg	-	-	60.56	15.078	1.603
PSNR (dB)		-	-	31.25	31.92	30.31

TABLE VI
Performance Analysis of Proposed Tree Technique using Raspberry Pi

		snapshot				
Image Name		1.bmp	2.bmp	3.bmp	4.bmp	5.bmp
Image Size (MB)		14.4	3.6	0.9	0.225	0.025
Compression Ratio		-	1.0414	1.0945	1.0979	1.0903
Compression Time (s)	Min	-	110.221	26.39	6.72	0.89
	Max	-	113.107	27.01	6.97	0.92
	Avg	-	111.99	26.71	6.86	0.903
Decompression Time (s)	Min	-	101.24	23.96	6.03	0.68
	Max	-	103.88	24.77	6.23	0.71
	Avg	-	102.325	24.39	6.12	0.699
PSNR (dB)		-	29.28	31.25	31.92	30.31

TABLE VII
Decompression Performance Comparison of the Proposed and Regular Huffman Techniques in Windows Platform

		Snapshot				
Source Name		1.bmp	2.bmp	3.bmp	4.bmp	5.bmp
Image Size(MB)		14.4	3.6	0.9	0.225	0.025
Time(s)	Regular Huffman (RH)	184.51	44.80	11.22	2.95	0.338
	Proposed Huffman (PH)	66.75	16.65	4.11	1.101	0.137
Enhancement Rate ((RH-PH)*100)/RH		63.82	62.83	63.36	62.67	59.46

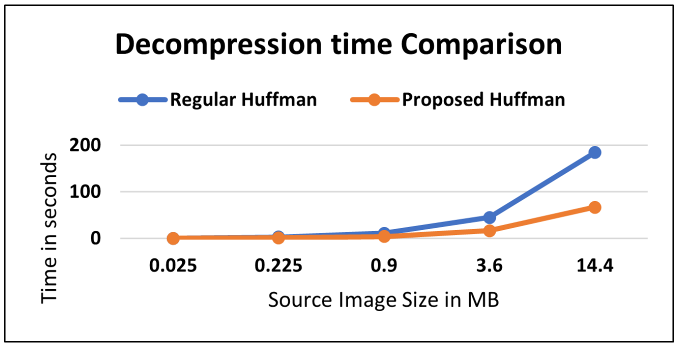


Fig. 5. Decoding Time comparison in Windows Platform

TABLE VIII
DECOMPRESSION PERFORMANCE COMPARISON OF THE PROPOSED AND REGULAR HUFFMAN TECHNIQUES IN RASPBERRY PI

		Snapshot				
Source Name		1.bmp	2.bmp	3.bmp	4.bmp	5.bmp
Image Size(MB)		14.4	3.6	0.9	0.225	0.025
Time(s)	Regular Huffman (RH)	-	-	60.56	15.078	1.603
	Proposed Huffman (PH)	-	-	24.39	6.12	0.699
Enhancement Rate ((RH-PH)*100)/RH		-	-	59.72	59.40	56.39

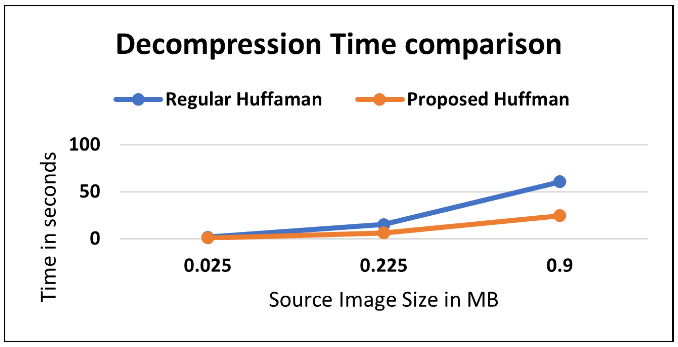


Fig. 6. Decoding Time comparison in Raspberry Platform

As the results of Figure 5 and Figure 6, the decompression speed of the newly proposed technique is higher than the existing regular Huffman-based technique for both Windows and Raspberry Pi platforms. In table VII, It has been shown that the proposed technique is more than 59% faster than the basic Huffman technique in terms of decoding time at the windows platform. In table VIII, it has been observed that the proposed method is more than 56% faster than the regular Huffman in the Raspbian platform. The proposed technique outperforms the regular Huffman-based technique in terms of decoding time as image size increases. Based on the findings, the proposed technique does not perform better in terms of compression ratio and compression time. Both approaches' average compression time is roughly identical.

The binary Huffman technique compresses slightly better than the proposed technique. As demonstrated , the proposed method is inefficient in terms of memory usage. It has been discovered that the proposed technique performs better in both platforms for all images except those with a large image size (14.4MB- snapshot1). Due to the poor processing capability of the Raspberry Pi Processor, the regular Huffman technique does not perform properly on images that are larger than 3.6 MB. In consequence, it has been evaluated that the proposed technique outperforms the consistent Huffman technique on low-processing-power machines.

V. CONCLUSION

The performance of constructing Huffman code using a newly proposed tree structure vs the conventional binary tree is contrasted in this paper. The research area focuses on minimizing decoding time to speed up processing. The newly proposed technique is compared to the existing Huffman-based algorithm. In comparison to employing a binary tree, the traversal time of a newly proposed tree structure is minimal. The proposed technique searches two bits at a time during the decoding process, which ensures a faster search than traditional linear search. Consequently, in terms of processing speed, the representation of Huffman code using a predicted tree structure is more advantageous than the traditional Huffman binary tree. The existing Huffman-based technique uses single-bit code to store data in memory, which slows decoding. By contrast, the newly proposed algorithm introduces a two-bit code to store data in memory. Therefore, the proposed Huffman tree structure performs better because it decodes two bits at a time from memory. The experiment concludes that the proposed technique is more time-efficient than the regular Huffman technique. The research topic can be expanded to include a balanced Quaternary tree, which can increase compression and decompression speeds simultaneously. In addition, the proposed technique may be extended for the video application data and 3D images as future work.

REFERENCES

[1] J. Oliver and M. P. Malumbres, "Huffman Coding of Wavelet Lower Trees for Very Fast Image Compression," in 2006 IEEE International Conference on Acoustics Speed and Signal Processing Proceedings, Toulouse, France, 2006, vol. 2, p. II-465-II–468. doi: 10.1109/ICASSP.2006.1660380.
[2] A. M. Rufai, G. Anbarjafari, and H. Demirel, "Lossy medical image compression using Huffman coding and singular value decomposition," in 2013 21st Signal Processing and Communications Applications Conference (SIU), Haspolat, Apr. 2013, pp. 1–4. doi: 10.1109/SIU.2013.6531592.
[3] A. Habib, M. J. Islam, and M. S. Rahman, "A dictionary-based text compression technique using quaternary code," Iran J. Comput. Sci., vol. 3, no. 3, pp. 127–136, Sep. 2020, doi: 10.1007/s42044-019-00047-w.
[4] R. A. Chowdhury, M. Kaykobad, and I. King, "An efficient decoding technique for Huffman codes," Inf. Process. Lett., vol. 81, no. 6, pp. 305–308, Mar. 2002, doi: 10.1016/S0020-0190(01)00243-5.
[5] C. Hong-Chung, W. Yue-Li, and L. Yu-Feng, "A memory-efficient and fast Huffman decoding algorithm," Inf. Process. Lett., vol. 69, no. 3, pp. 119–122, Feb. 1999, doi: 10.1016/S0020-0190(99)00002-2.
[6] D. A. Huffman, "A Method for the Construction of Minimum-Redundancy Codes," p. 4.
[7] K.-L. Chung, "Efficient Huffman decoding," p. 3, 1997.

[8] R. Hashemian, “Memory efficient and high-speed search Huffman coding,” IEEE Trans. Commun., vol. 43, no. 10, pp. 2576–2581, Oct. 1995, doi: 10.1109/26.469442.

[9] P. R. Suri and M. Goel, “Ternary Tree and Memory-Efficient Huffman Decoding Algorithm,” vol. 8, no. 1, p. 7, 2011.

[10] Y.-K. Lin, S.-C. Huang, and C.-H. Yang, “A fast algorithm for Huffman decoding based on a recursion Huffman tree,” J. Syst. Softw., vol. 85, no. 4, pp. 974–980, Apr. 2012, doi: 10.1016/j.jss.2011.11.1019.

[11] X. Li and Y. Shen, “A Medical Image Compression Scheme Based on Low Order Linear Predictor and Most-likely Magnitude Huffman Code,” in 2006 International Conference on Mechatronics and Automation, Luoyang, Jun. 2006, pp. 1796–1800. doi: 10.1109/ICMA.2006.257487.

[12] R. Ranjan, “Canonical Huffman Coding Based Image Compression using Wavelet,” Wirel. Pers. Commun., vol. 117, no. 3, pp. 2193–2206, Apr. 2021, doi: 10.1007/s11277-020-07967-y.

[13] S. Thummala, “FPGA Implementation of Huffman Encoder and Decoder for High Performance Data Transmission,” Int. J. Eng. Res., vol. 3, no. 2, p. 5, 2014.

[14] Fruchtman, Aharon Gross, Yoav Klein, Shmuel Shapira, Dana. (2020). Weighted Adaptive Coding.

[15] J. Matai, J.-Y. Kim, and R. Kastner, “Energy efficient canonical huffman encoding,” in 2014 IEEE 25th International Conference on Application-Specific Systems, Architectures and Processors, Zurich, Switzerland, Jun. 2014, pp. 202–209. doi: 10.1109/ASAP.2014.6868663.