Does Deflate Outperform Huffman Lossless Compression?

1st Vincent Dava Sutomo
Binus University
Tanggerang, Indonesia
vincentdavas@gmail.com

2nd Neo Cenon
Binus University
Tanggerang, Indonesia
neocenon@gmail.com

3rd Aloysius Kang
Binus University
Tanggerang, Indonesia
aloysiuskang123@gmail.com

4th I Dewa Agung Kayana Abhipraya Putra Bandjar
Binus University
Tanggerang, Indonesia
kayanaabhipraya041@gmail.com

***Abstract*—Lossless compression is a compression technique used to store data with higher quality compared to its counterpart, lossy compression. This study explores two specific lossless compression techniques to investigate and address the research questions that motivated this experiment. Deflate, an algorithm derived from Huffman and LZ77, is one of the techniques under examination. This algorithm is particularly intriguing due to the amalgamation of the two preceding compression techniques. Consequently, a crucial inquiry arises: does Deflate surpass its original algorithm? In this paper, Huffman and Deflate are compared to analyze whether combining algorithms yields superior or inferior results. The conducted test provides evidence supporting the superiority of Deflate over Huffman. Deflate demonstrates advantages in terms of compression ratio and, notably, compression and decompression speed. However, it is important to note that this case is not conclusively closed, as the experiment can still be extended to include other algorithms. The present paper exclusively focuses on Deflate and does not guarantee that all combination-based algorithms outperform the algorithms they incorporate. Nevertheless, this comparative analysis makes a valuable contribution to the future development of lossless compression techniques. Additionally, it fulfills the objective of understanding the factors that enable an algorithm like Deflate to effectively implement multiple compression techniques.**

***Keywords*—lossless compression, Huffman code, Deflate**

I. Introduction

Since the invention of Morse code in 1838, data compression has been the subject of extensive study. There have been further developments in modern data compression techniques since then. Claude Shannon and Robert Fano developed a mechanism for assigning codewords based on block probabilities around 1949. As technology has become more significant and prevalent, the field has continued to advance steadily. Notable milestones include the development of Huffman coding in the 1970s [**?**], the LZW Algorithm in the 1980s [1], the Lossy algorithm in the early 1990s, and the lossless algorithm in the 2000s.

Lossless compression attempts to reduce the file size without sacrificing quality [2]. This technique has numerous applications, including databases, spreadsheets, images, and videos [3]. Lossless compression has undergone numerous iterations since its inception in an effort to improve and perfect the concept.

Nevertheless, there are numerous compression techniques, each with its own set of benefits and drawbacks. This can make it difficult to determine the most appropriate technique for various situations. Consequently, it has resulted in the emergence of papers that compare and analyze algorithms using various methods. The title of one such paper is ”Modern Lossless Compression Techniques: Review, Comparison, and Analysis.” It investigates the use of run-length encoding and Huffman encoding in contemporary compression techniques such as LZ-77 Encoding, Deflate, and LZMA. This paper evaluates the compression ratio, compression speed, and decompression speed of algorithms [4].

Another paper, ”Comparison of Lossless Data Compression Techniques,” examines lossless compression techniques, including Delta encoding, Lempel-Ziv-Welch, Huffman encoding, and Run-Length encoding. Their evaluation concludes that Lempel-Ziv-Welch has the greatest overall performance among the methods that have been examined. The compression ratio of Lempel-Ziv-Welch is approximately 4:1, resulting in a space savings of 76.9% [5].

In the third paper, titled ”Lossless Data Compression Techniques and Their Performance,” three compression techniques are investigated: Huffman coding, LZW, and Shannon-Fann coding. Instead of determining the optimal algorithm, it evaluates the performance of each algorithm based on certain parameters. Specifically, it describes how Huffman coding achieves the lowest compression ratio, whereas LZW achieves the maximum compression ratio [6].

The purpose of this study is to evaluate and analyze various lossless compression techniques. Among the algorithms being analyzed are well-known ones like LZ-77, LZ-SS, and Huffman encoding [7]. To ensure consistency and comprehensiveness, a fixed dataset will be used for all algorithm evaluations, and multiple comparison parameters will be employed. This includes compression ratio, compression rate, and decompression rate. This study will provide a comprehensive and up-to-date comparison of commonly employed compression techniques [4].

This study will commence by providing a summary of the consensus reached in each paper regarding lossless compression algorithms. This will be achieved through the use of an annotated bibliography and metrics to identify the methods, results, and discrepancies in each paper. In conclusion, the paper will describe the entire procedure, beginning with the preprocessing phase and concluding with the evaluation of lossless compression algorithms [8].

This article is divided into five separate sections. Section II provides previous research of losless algorithm. The methodology of the study is elaborated in the third section, providing a comprehensive account of the procedures and methodologies employed to scrutinize the dataset and produce the outcomes. Section IV presents a comprehensive overview of the findings and their implications, as revealed by the analysis. The final section, Section V, provides a concise overview of the primary discoveries of the investigation and an analysis.

II. Previous Research

[4] discusses the main distinctions between data compression techniques used to reduce the size of data for storage and transmission over channels with limited bandwidth. There are two categories of compression techniques: lossless and lossy. Both strive to reduce file size, allowing for efficient file delivery with minimal bandwidth consumption. Lossless compression techniques use Huffman coding to reduce the number of bits in a file without sacrificing any data. In contrast, lossy image and video compression techniques approximate clusters of pixels into a single value for comparison purposes. The algorithms' output results are evaluated based on their compression ratio and speed.

In their study, Alyami et al. [9] compared compression algorithms such as Nongreedy unbalanced Huffman tree (NUHT), Unbalanced Huffman tree (UHT), gzip, bzip2, and specialized compressors for genomic data and DNA sequences. The objective was to assess the efficiency of these algorithms, focusing on compression ratio and speed. The study's findings led to the conclusion that the NUHT algorithm maximizes the use of 2-mers, as its impact factor ratio was the highest. By choosing the optimal 2-mer, the algorithm effectively minimized the base 4 count and built an optimal Huffman tree. Compared to gzip and bzip2, as well as conventional Huffman tree encoding, this method produced a superior compression ratio. In terms of performance and compression ratio, NUHT exhibited minor improvements over UHT's fundamental algorithm.

Devi and Kotha [8] analyze the significance of data compression for conserving storage space, facilitating quick file transfers, and optimizing storage bandwidth. Lossy compression and lossless compression are the two primary types of data compression methods classified by the authors. This publication focuses predominantly on lossless compression techniques, such as Huffman coding and Arithmetic coding.

Following our comprehensive review of relevant literature, we proceeded to undertake a comparative analysis of the Deflate algorithm and Huffman coding.

III. Methodology

A. Data

Our paper focuses on testing and doing comparative analysis over two algorithms which are Huffman and Deflate. Therefore, the number of images tested is not a lot. For the sake of demonstration, we have three google images in BMP format. BMP is suitable for the occasion because of its larger file size and the high-quality image that it can store. Also, the three images all have different sizes ranging from small to large. Details for the image size, resolution is in Fig 1.

B. Huffman Code

Huffman Coding is one of the lossless techniques that can be used to compress a file in order to not losing any data, in this review we will discuss more specifically to the image. image itself has 256 levels of gray and colours. Huffman method is included in statistical approach in image compression, which means the color or degree of gray that often appears in in the image will be encoded with the number fewer bits while the value is the frequency of occurrence is slightly coded with a longer number of bits.

Huffman algorithm to image compression [5]:

1) Create image data in the form of a matrix into a vector.
2) Determine the frequency of occurrence of each color or degree of gray.
3) Sort by color ascending or gray value based on frequency occurrence or opportunity occurrence of pixels in the image. Each values are expressed as trees single knot and every node assigned with frequency appearance of this value.
4) Then combine 2 pieces of that tree have an occurrence frequency smallest at a root. Root has a frequency which is the sum of the frequencies of the two trees constituent. Frequency with smaller value is placed on the left side and is given a weight of 0 while the right side given a weight of 1.
5) Repeat step 4 until only 1 binary tree remains.
6) Trace the binary tree from root to leaf. Sequence side weights from root to leaves represent Huffman codes for the degree of gray or color appropriate
7) Replacing data with Corresponding Huffman code.
8) Store image width, height data image, bit code for each value, colors data that contained in the image, and encoded image data into the compressed file

We also need decompressed algorithm to restore an image compressed into the original image, decompressed algorithm itself is the reverse part of compression algorithm. This is how decompressed algorithm works:

1) Firstly, read the compressed file and input the data into the corresponding variable which are size image variable, data bit code variable, and colors variable
2) Then, read the data code bit by bit from left to right and matched against the Huffman code of data color that we got. And so on conversion carried out until the last of the data.
3) Lastly, Reconstruct the image with using image size data, which means the pixel data is in 1D form decapitated rows and columns accordingly to the image size.

C. Deflated Compression Algorithm

Deflate is a lossless data compression algorithm that uses a combination of the LZ77 algorithm and Huffman coding algorithm. Deflate has widespread use, mainly for image processing and zip files. Deflate uses a chained hash table to find duplicated strings, using a hash function that operates on typically 2 or 3-byte sequences [6]. It finds duplicated strings in the input data. The second occurrence of a string is replaced by a pointer to the previous string, in the form of a pair (distance, length). Distances are limited to 32K bytes, and lengths are limited to 258 bytes. When a string does not

(a) Small Image: 640 * 480, 900 KB

(b) Medium Image: 800 * 600, 1.4 MB

(c) Large Image: 2292 * 1500, 9.8 MB

Fig. 1: Dataset

occur anywhere in the previous 32K bytes, it is emitted as a sequence of literal bytes [5].

D. Evaluation

There are 3 parameters measured between uncompressed image and compressed image that we use:

1) Compression Ratio
2) Compression Speed
3) Decompression Speed

Compression Ratio is the ratio of uncompressed image size to the compressed image size. The lower the compression ratio, the bigger the compressed image size. Compression ratio can indicate the efficiency of compression algorithms [10].

$$compressionration(CR) = \frac{originalimagesize}{compressedimagesize} \quad (1)$$

Compression speed is a parameter that measures how fast the compression algorithm makes the image file compressed. Compression speed measured in MB/s of input data consumed (uncompressed data handled per second). Compression speed refers to how quickly the compression algorithm is performed. Different compression algorithms have different complexity and efficiency [5].

Decompression speed is a parameter that measures how fast the decompression algorithm make the image file uncompressed. Decompression speed measured in MB/s for decompressed file. Decompression speed refers to how quickly the decompression algorithm is performed.

IV. Experiment and Discussion

In this paper, the result will be based on the three parameters mentioned above. Compression ratio, compression speed, and decompression speed. All three parameters will be checked and tested on different sizes to see the changes between results and to have a more concrete evaluation later on about the two algorithms compared. The library used for Huffman is daHuffman, https://pypi.org/project/dahuffman/. The library used for Deflate is ZLib, https://docs.python.org/3/library/zlib.html. As mentioned above, we were using three image dataset which is small, medium and large resolution image.

We are using CPU time and Wall time for compression speed and decompression speed. CPU time measures how much time the CPU was busy executing the code. Wall time measures how much time in the real world has passed between method entry and method exit. For the implementation, we use ‘import time’ in python to know the compression speed and decompression speed in python. We put ”%%time’ in the block of code that runs the compression/decompression algorithm until it finishes compressing/decompressing to measure the CPU Time and Wall Time.

Table 1 represent the result based on compression ratio and Table 2 represent the result based on Compression Time and Decompression Time from our data.

TABLE I: Compression Ratio from our dataset

Algorithm	Small	Medium	Large
Huffman Code	1.0426	1.01097	1.01595
Deflate	1.19746	1.21838	1.09161

The tables provided display the results of an analysis performed on compression ratios, compression velocities, and decompression speeds for the Huffman Code and Deflate algorithms.

The compression ratios for small, medium, and large data inputs are presented in Table I. It demonstrates that Huffman Code achieves compression ratios between 1.01097 and 1.0426, whereas Deflate achieves compression ratios between 1.09161 and 1.21838.

The compression and decompression rates based on CPU time are presented in Table II. For each algorithm and input data size (small, medium, or large), the compression and decompression velocities are specified. Notably, Deflate compresses and decompresses data substantially faster than Huffman Code in all circumstances.

The compression and decompression rates based on wall time are presented in Table III. Similar to Table II, Deflate compression and decompression speeds are speedier than those of Huffman Code for all data input sizes.

Overall, these tables illustrate the differences in compression ratios, compression velocities, and decompression speeds between Huffman Code and Deflate. Deflate consistently demonstrates greater compression ratios and compres-

TABLE II: Compression Speed and Decompression Speed based on CPU time from our dataset

Algorithm	Small		Medium		Large	
	Compression Speed	Decompression Speed	Compression Speed	Decompression Speed	Compression Speed	Decompression Speed
Huffman Code	351 ms	1.67 s	348 ms	2.1 s	2.54 s	17 s
Deflate	42.3 ms	9.7 ms	49.3 ms	8.99 ms	361 ms	56.9 ms

TABLE III: Compression Speed and Decompression Speed based on Wall time from our dataset

Algorithm	Small		Medium		Large	
	Compression Speed	Decompression Speed	Compression Speed	Decompression Speed	Compression Speed	Decompression Speed
Huffman Code	360 ms	1.84 s	351 ms	2.11 s	2.56 s	17.2 s
Deflate	40.7 ms	9.73 ms	52.1 ms	11.5 ms	361 ms	57.2 ms

sion/decompression velocities than Huffman Code, indicating its potential superiority in certain applications.

V. Conclusion

In this paper, a comparative analysis was conducted between two lossless compression algorithms: Huffman and Deflate. The experiment focused on evaluating compression ratios, compression speeds, and decompression speeds from three different size images.The results indicated that Deflate outperformed Huffman in terms of compression ratio, with a ratio ranging from 1.09161 to 1.21838, while huffman's ratio ranging between 1.01097 and 1.0426. The superiority of Deflate was further evident in both compression and decompression speeds, where it demonstrated significantly faster performance compared to Huffman.

Based on these results, it can be concluded that Deflate, derived from Huffman and LZ77, surpasses its original algorithm in terms of performance. However, it is worth noting that further analysis is required to explore additional parameters that were not the focus of this paper.

References

[1] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Transactions on Information Theory*, vol. 23, no. 3, pp. 337–343, May 1977. [Online]. Available: http://ieeexplore.ieee.org/document/1055714/

[2] A. Hussain, A. Al-Fayadh, and N. Radi, "Image compression techniques: A survey in lossless and lossy algorithms," *Neurocomputing*, vol. 300, pp. 44–69, Jul. 2018. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0925231218302935

[3] M. Ledwon, B. F. Cockburn, and J. Han, "High-Throughput FPGA-Based Hardware Accelerators for Deflate Compression and Decompression Using High-Level Synthesis," *IEEE Access*, vol. 8, pp. 62 207–62 217, 2020. [Online]. Available: https://ieeexplore.ieee.org/document/9050498/

[4] A. Gupta, A. Bansal, and V. Khanduja, "Modern lossless compression techniques: Review, comparison and analysis," in *2017 Second International Conference on Electrical, Computer and Communication Technologies (ICECCT)*. Coimbatore: IEEE, Feb. 2017, pp. 1–8. [Online]. Available: http://ieeexplore.ieee.org/document/8117850/

[5] A. Gopinath and M. Ravisankar, "Comparison of Lossless Data Compression Techniques," in *2020 International Conference on Inventive Computation Technologies (ICICT)*. Coimbatore, India: IEEE, Feb. 2020, pp. 628–633. [Online]. Available: https://ieeexplore.ieee.org/document/9112516/

[6] K. Sharma and K. Gupta, "Lossless data compression techniques and their performance," in *2017 International Conference on Computing, Communication and Automation (ICCCA)*. Greater Noida: IEEE, May 2017, pp. 256–261. [Online]. Available: http://ieeexplore.ieee.org/document/8229810/

[7] K. Yao, H. Li, W. Shang, and A. E. Hassan, "A study of the performance of general compressors on log files," *Empirical Software Engineering*, vol. 25, no. 5, pp. 3043–3085, Sep. 2020. [Online]. Available: https://link.springer.com/10.1007/s10664-020-09822-x

[8] H. Devi Kotha, M. Tummanapally, and V. K. Upadhyay, "Review on Lossless Compression Techniques," *Journal of Physics: Conference Series*, vol. 1228, no. 1, p. 012007, May 2019. [Online]. Available: https://iopscience.iop.org/article/10.1088/1742-6596/1228/1/012007

[9] S. Alyami and C.-H. Huang, "Nongreedy Unbalanced Huffman Tree Compressor for Single and Multifasta Files," *Journal of Computational Biology*, vol. 27, no. 6, pp. 868–876, Jun. 2020. [Online]. Available: https://www.liebertpub.com/doi/10.1089/cmb.2019.0249

[10] R. Patel, V. Kumar, V. Tyagi, and V. Asthana, "A fast and improved Image Compression technique using Huffman coding," in *2016 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*. Chennai, India: IEEE, Mar. 2016, pp. 2283–2286. [Online]. Available: http://ieeexplore.ieee.org/document/7566549/