ISSN: 2414-1895 DOI: 10.6919/ICJE.202512_11(12).0017

Analysis of Huffman Tree-Based Lossless Compression on Different Text Types

Lanzhi Luo, Qinyan Yu, Heng Zhang, Shengqi Pan

Shanghai United International School (Qingpu Campus), Qingpu, Shanghai, 201704, China

Abstract

This paper comprehensively investigates Huffman Tree-based lossless data compression across various text types, including technical reports, news articles, and fiction narratives. A detailed theoretical model for calculating compression ratios is first introduced and subsequently validated through systematic MATLAB simulations. The results clearly indicate that character frequency distribution significantly affects compression efficiency. Specifically, fiction narrative, with its higher redundancy and repetitive linguistic patterns, achieves the best performance, yielding the highest compression ratio. Conversely, more diverse and concise texts like technical documents show less dramatic gains. These findings robustly demonstrate Huffman coding's particular effectiveness and efficiency when applied to high-redundancy datasets, reinforcing its foundational role in classical compression theory.

Keywords

Huffman Tree; Lossless Compression; MATLAB Simulation.

1. Introduction

In the contemporary digital era, where vast volumes of data—ranging from text and images to video and audio—are continuously generated, a fundamental consideration must be emphasized: the efficient storage and transmission of information are as critical as its creation [1-5]. Without proper compression and encoding techniques, the burden on storage systems and communication channels would become unsustainable, leading to latency, cost, and scalability issues. This recognition highlights a key concept in computer science: the Huffman Tree. Proposed by David A. Huffman in 1952, this tree-based algorithm fundamentally transformed the methodology of lossless data compression [6], offering a systematic approach to minimize redundancy without sacrificing information integrity. It has since been widely adopted in numerous applications, including file compression formats such as ZIP and image standards like JPEG [7].

The Huffman Tree is not merely an elegant data structure, but also a highly efficient compression strategy grounded in a simple but powerful principle: elements with higher frequencies of occurrence should be represented by shorter codes, whereas less frequent elements are assigned longer codes. This principle directly leverages the statistical properties of source data to reduce average code length, thereby achieving high compression ratios [8]. The efficacy of the algorithm lies not only in its conceptual foundation but also in its systematic implementation. Constructed in a bottom-up manner, the tree begins with nodes representing the smallest weights. Then, it incrementally builds a hierarchical structure where the most frequently occurring symbols occupy the shortest paths. The outcome is a prefix-free binary code, ensuring that no encoded symbol is a prefix of another, thereby guaranteeing unambiguous decoding and enabling fast, reliable decompression in real-time systems [9].

This paper investigates the theoretical foundation, practical application, and compression performance of the Huffman Tree. The remainder of this paper is structured as follows. Section 2

ISSN: 2414-1895 DOI: 10.6919/ICJE.202512_11(12).0017

introduces the model and algorithm structure. Section 3 simulates the Huffman coding on various text types and evaluates the compression performance. Section 4 concludes the findings of this work.

2. Models

The model used for data compression in this work is the Huffman Tree. It is an algorithm that approaches the theoretical limit (entropy) of data compression, which is given by [10]:

$$L = -\sum p(x_i) \log_2 p(x_i) \tag{1}$$

In constructing a Huffman Tree, we let the path length be l_i, representing the distance from the root to a specific node, and ω_i be the frequency of the node. The Huffman Tree model achieves the minimum weighted path length (WPL) of a binary tree [10]. The WPL is defined as:

$$WPL = \sum l_i \omega_i \tag{2}$$

Two samples of trees for calculating WPL are shown in Fig. 1. The WPL is calculated as follows: 3×1+3×2+2×3+1×4=19 for Fig. 1(a) and 1×1+2×2+3×3+3×4=26 for Fig. 1(b).

To construct a Huffman Tree, the sequence of frequencies is first arranged from largest to smallest. These frequencies are treated as initial nodes. The two nodes with the smallest frequencies (child nodes) are selected and merged into a parent node whose frequency is the sum of the two. This new parent node is reinserted into the list, replacing its child nodes. The sequence is then re-sorted, and the process is repeated until only a single root node remains. Because each parent node is formed by combining two child nodes, the process must be repeated *n-1* times for *n* initial nodes. This results in the creation of *n-1* new nodes, leading to a total of *2n−1* nodes in the final tree. To apply the Huffman Tree for data compression, each branch in the tree is represented by a binary digit-'0' for the smaller frequency node and '1' for the larger. The encoding of a symbol is the binary sequence traced from the root to the corresponding initial node.

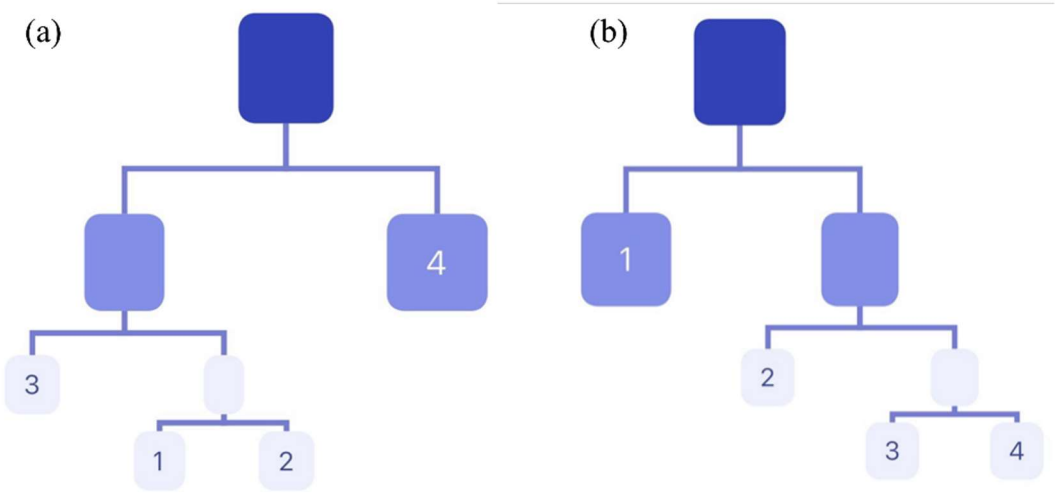

Fig. 1 Tree samples for calculating WPL (The numbers in the boxes represent the frequency). (a) Sample 1 (b) Sample 2

In Fig. 2, a Huffman Tree example is constructed for the frequencies (2, 3, 4, 8, 9). The WPL is calculated as 3×2+3×3+2×4+2×8+2×9=57.

ISSN: 2414-1895 DOI: 10.6919/ICJE.202512_11(12).0017

The encoding for each symbol is listed in Table 1. Originally, these symbols would require 2+3+4+8+9=26 bytes in a commercial computer, equal to 26×8=208 bits. After Huffman compression, only 57 bits are needed. The compression ratio is therefore 57/208=27.4%.

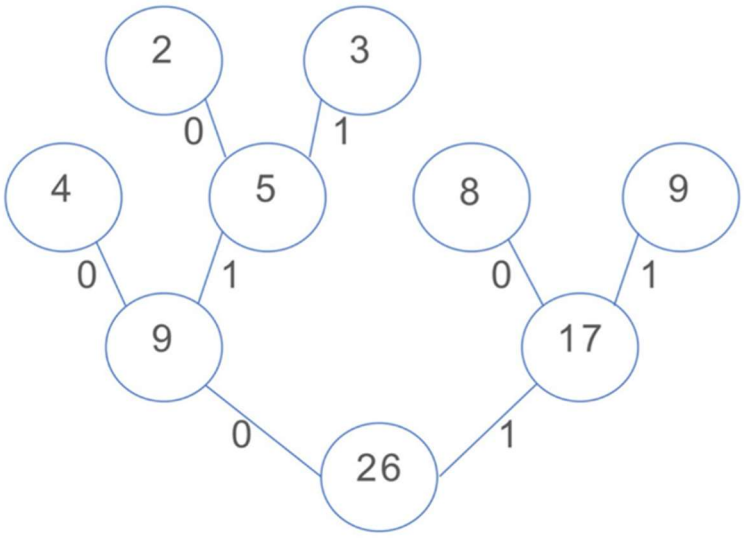

Fig. 2 A Huffman Tree example

Table 1. Huffman code for initial nodes in Fig. 2

Initial nodes	2	3	4	8	9
code	010	011	00	10	11

3. Simulation Results

Based on the Huffman encoding algorithm, the compression performance across different types of input text was investigated. The text types and their original file size are listed in Table 2. All simulations were conducted in MATLAB 2020 to maintain consistency.

Table 2. Original text type and file size

Text type	Friction narrative	Poetry	Technical writing
Original file size	4424 bits	2944 bits	5968 bits

The content of the original files was read into a string using plain extended ASCII encoding, ensuring consistent representation of characters across different text types. A one-dimensional array was used to record the frequency of each unique character. A struct array was then created to store Huffman Tree nodes, including the character (for initial nodes), its frequency, and pointers to child nodes. The Huffman Tree was constructed by repeatedly merging the two least frequent nodes using an ascending sort algorithm until only the root node remained. Then, a binary string representing each unique symbol was then generated by traversing from initial nodes to the root using nested loops and selection statements. The input text was subsequently converted into a binary string using these symbol encodings, and the compression ratio was calculated by comparing the size of the encoded bitstream to the original input size. A lower compression ratio value indicates better compression performance.

ISSN: 2414-1895 DOI: 10.6919/ICJE.202512_11(12).0017

The simulation results are shown in Table 3 and Fig. 3. Friction Narrative achieves the highest performance with a CR of 51.20%. This is because of its lowest entropy of 4.06 bits per character due to heavy repetition of phases, frequent use of high-frequency characters (‘ ’, 'e', 't'), standard grammar and predictable transitions, where a few characters dominate the distribution. These features lead to an unbalanced frequency distribution, which Huffman coding leverages effectively. In contrast, Technical Writing yields the lowest performance with CR 54.56% and the highest entropy of 4.34 bits per character. This is primarily due to its specialised vocabulary, weak symbol frequency skew and minimal redundancy in phasing. Poetry exhibits intermediate compression efficiency with moderate entropy and moderate CR values. The data result indicates that the Huffman encoding algorithm is particularly suitable for datasets with skewed character frequency distributions. The input text with a higher entropy tends to achieve lower potential for compression as very short codes cannot be assigned to many symbols due to a flatter distribution. This reinforces the principle that the effectiveness of Huffman coding methods is tightly linked to the structure of the input data.

Table 3. Compressed file size and ratio

Text type	Friction narrative	Poetry	Technical writing
Compressed file size	2265 bits	1605 bits	3256 bits
Compression ratio	51.20%	54.52%	54.56%

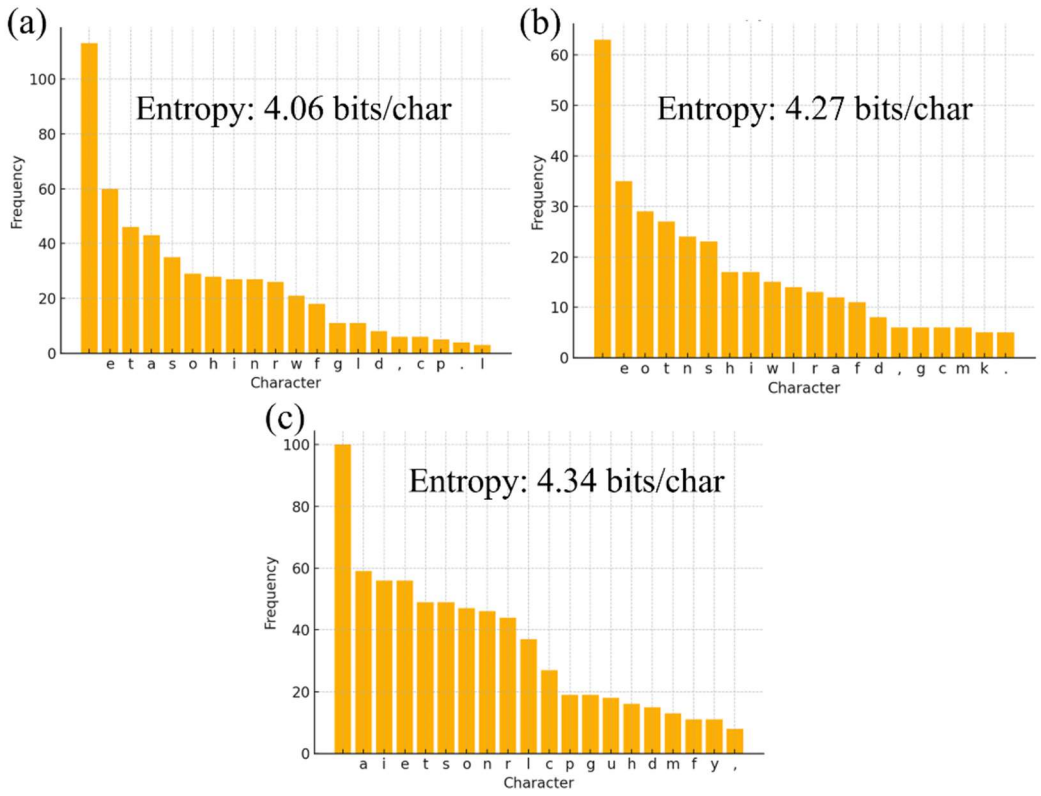

Fig. 3 Entropy and character frequency distribution for each input text type (a) Friction Narrative (b) Poetry (c) Technical Writing

4. Conclusion

This paper presents a comprehensive analysis of Huffman Tree-based compression, with a focus on how textual characteristics influence compression efficiency. A Huffman model is first introduced

ISSN: 2414-1895 DOI: 10.6919/ICJE.202512_11(12).0017

through several examples. Subsequently, simulations were conducted in MATLAB to evaluate the performance of the Huffman algorithm across three types of text: fiction narrative, poetry, and technical writing. The results indicate that the distribution of character frequencies plays a critical role in determining the compression ratio. Among the evaluated categories, fiction narrative achieved the highest compression efficiency, with a compression ratio of 51.20%, reducing the file size from 4424 bits to 2265 bits. This outcome highlights the strong performance of Huffman encoding on text types characterized by relatively high redundancy. Overall, the findings confirm the practical applicability of Huffman Tree coding for lossless data compression and offer insights into its effectiveness across diverse linguistic structures.

References

[1] D. A. Lelewer and D. S. Hirschberg, "Data compression," ACM Computing Surveys (CSUR), vol. 19, no. 3, pp. 261–296, 1987.

[2] J. Chen, Y. Fang, A. Khisti, A. Özgür, and N. Shlezinger, "Information compression in the AI era: Recent advances and future challenges," IEEE Journal on Selected Areas in Communications, 2025.

[3] F. Liu et al., "Ultralow photon flux OWC links using UV-C single-photon detection," Laser & Photonics Reviews, p. 2401804, 2025.

[4] F. Liu et al., "Ultra-sensitive UV solar-blind optical wireless communications with an SiPM," Optics Letters, vol. 48, no. 20, pp. 5387–5390, 2023/10/15 2023, doi: 10.1364/OL.503235.

[5] F. Liu et al., "10 Mbit/s UV Solar-Blind OWC at 30 Photons Per Bit," in 2024 Conference on Lasers and Electro-Optics (CLEO), 5–10 May 2024 2024, pp. 1–2.

[6] D. A. Huffman, "A method for the construction of minimum-redundancy codes," Proceedings of the IRE, vol. 40, no. 9, pp. 1098–1101, 2007.

[7] L. A. Fitriya, T. W. Purboyo, and A. L. Prasasti, "A review of data compression techniques," International Journal of Applied Engineering Research, vol. 12, no. 19, pp. 8956–8963, 2017.

[8] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, "Introduction to Algorithms (3-rd edition)," MIT Press and McGraw-Hill, 2009.

[9] D. Salomon and G. Motta, Handbook of data compression. Springer Science & Business Media, 2010.

[10] C. E. Shannon, "A mathematical theory of communication," The Bell System Technical Journal, vol. 27, no. 3, pp. 379–423, 1948, doi: 10.1002/j.1538-7305.1948.tb01338.x.