ISSN No : -2456-2165 https://doi.org/10.38124/ijisrt/26may1486

Data Compression Using Huffman Coding & Decoding in MATLAB

Pulak Kumar Jena[1]; Ramkumar Ghadai[2]; Sudarshan Murmu[3]; Binod Kumar Baliar Sing[4]

[1,2,3,4] Konark Institute of Science & Technology , Jatani Khurda

Publication Date: 2026/06/04

Abstract: The rapid advancement of digital communication systems and multimedia technologies has significantly increased the amount of data generated, transmitted, and stored across computer networks. As a result, efficient data compression techniques have become essential for minimizing storage requirements and reducing transmission bandwidth while maintaining data integrity. Data compression is a process that eliminates redundancy from digital information to represent data using fewer bits than the original format. Among various lossless compression techniques, Huffman Coding remains one of the most effective and widely adopted methods due to its simplicity, optimality, and efficient implementation characteristics.

Huffman Coding is a statistical compression technique that generates variable-length binary codes based on the probability of occurrence of symbols within a dataset. Frequently occurring symbols are assigned shorter binary codes, while less frequent symbols receive longer codes, thereby reducing the average code length of the encoded data. This approach achieves efficient compression without any loss of information, making it suitable for applications where exact reconstruction of original data is required. Huffman Coding has been extensively used in file compression systems, text processing applications, image compression standards, and digital communication systems [1].

This research paper presents the implementation and analysis of Huffman Coding and Decoding using MATLAB. MATLAB provides a powerful computational environment for simulating digital communication algorithms and analyzing compression performance. The proposed work focuses on constructing Huffman trees based on symbol probabilities, generating optimal prefix codes, encoding input data into compressed binary sequences, and reconstructing the original information through the decoding process. The MATLAB implementation demonstrates the practical realization of Huffman Coding and evaluates the effectiveness of the algorithm using different performance parameters such as compression ratio, average code length, coding efficiency, and redundancy.

The research further analyzes how symbol probability distribution affects the overall compression performance. Experimental results indicate that Huffman Coding achieves higher compression efficiency when symbol frequencies are non-uniform. The encoded output generated through MATLAB simulation significantly reduces data size compared to fixed-length coding schemes while ensuring accurate recovery of the original message during decoding. The lossless nature of the algorithm makes it highly reliable for applications involving sensitive textual, image, and multimedia data [2].

In addition to implementation and performance evaluation, this paper discusses the advantages, limitations, and practical applications of Huffman Coding in modern communication systems. Although several advanced compression methods such as Arithmetic Coding and Lempel-Ziv-Welch (LZW) have been developed, Huffman Coding continues to remain a fundamental technique because of its lower computational complexity and efficient real-time performance. The study also highlights possible future improvements including adaptive Huffman Coding and hybrid compression methods integrated with intelligent algorithms [3].

The overall objective of this work is to provide a detailed understanding of lossless data compression using Huffman Coding and to demonstrate its practical implementation in MATLAB for educational and research purposes. The proposed system proves that Huffman Coding remains an effective solution for minimizing data storage requirements and improving communication efficiency in digital systems.

Keywords: *Data Compression, Huffman Coding, MATLAB, Lossless Compression, Encoding, Decoding, Compression Ratio, Information Theory.*

ISSN No : -2456-2165 https://doi.org/10.38124/ijisrt/26may1486

How to Cite: Pulak Kumar Jena; Ramkumar Ghadai; Sudarshan Murmu; Binod Kumar Baliar Sing (2026) Data Compression Using Huffman Coding & Decoding in MATLAB. *International Journal of Innovative Science and Research Technology*,11(5), 3057-3065. https://doi.org/10.38124/ijisrt/26may1486

I. INTRODUCTION

The development of modern digital technologies has resulted in a tremendous increase in the volume of data exchanged through communication networks and stored in electronic devices. Text files, audio signals, images, videos, and multimedia applications require large memory capacity and high transmission bandwidth for efficient processing and communication. The continuous growth of internet services, cloud computing, and multimedia streaming has further intensified the demand for efficient data storage and transmission systems. Consequently, data compression has become one of the most important research areas in computer science, digital communication, and information technology [1].

Data compression refers to the process of reducing the size of digital information by eliminating statistical redundancy while preserving the essential content of the original data. Compression techniques aim to minimize the number of bits required to represent information, thereby reducing storage space and communication bandwidth. In general, data compression techniques are classified into two major categories: lossless compression and lossy compression. Lossless compression techniques preserve the exact original information after decompression, whereas lossy compression methods permanently remove certain information to achieve higher compression ratios [2].

Lossless compression is particularly important in applications where data accuracy and integrity are critical. Examples include text documents, executable programs, medical images, scientific datasets, and database systems. Huffman Coding is one of the most widely used lossless compression techniques due to its ability to generate optimal variable-length prefix codes based on symbol occurrence probabilities. The algorithm was introduced by David A. Huffman in 1952 as a method for constructing minimum redundancy codes [3]. Since then, Huffman Coding has become a fundamental component of numerous compression standards and communication protocols.

The basic principle of Huffman Coding is based on assigning shorter binary codes to symbols that occur more frequently and longer codes to symbols with lower frequencies. This variable-length coding mechanism significantly reduces the average number of bits required to represent data. Unlike fixed-length coding schemes where each symbol is represented using an equal number of bits, Huffman Coding dynamically allocates code lengths according to symbol probabilities. As a result, the algorithm achieves improved coding efficiency and better compression performance [4].

Huffman Coding uses a binary tree structure called the Huffman tree for generating optimal prefix codes. The algorithm begins by calculating the frequency or probability of each symbol in the input dataset. Symbols are then arranged in ascending order of probability, and the two least probable symbols are repeatedly combined to construct the tree structure. Binary values are assigned to the branches of the tree, typically using ‘0’ for the left branch and ‘1’ for the right branch. The path from the root node to a symbol node determines the binary code assigned to that symbol [5]. The generated codes satisfy the prefix property, meaning that no codeword is the prefix of another codeword, thereby ensuring unambiguous decoding.

In recent years, MATLAB has emerged as a highly effective software platform for implementing and analyzing communication system algorithms. MATLAB provides powerful computational tools, matrix processing capabilities, graphical visualization functions, and built-in communication libraries that simplify the simulation of data compression techniques. Therefore, MATLAB is widely used in academic research and engineering applications for studying encoding and decoding algorithms [6]. In this research work, MATLAB is utilized to implement Huffman Encoding and Decoding processes and to evaluate the compression performance of the algorithm using different input datasets.

The proposed work aims to study the theoretical concepts of Huffman Coding and demonstrate its practical implementation using MATLAB simulation. The encoding process converts the original data into compressed binary sequences based on generated Huffman codes, while the decoding process reconstructs the exact original information from the compressed data. Performance analysis is carried out using parameters such as compression ratio, average code length, coding efficiency, and redundancy. These parameters help in evaluating the effectiveness of the proposed compression system [7].

The importance of Huffman Coding can be observed in several practical applications. It is commonly used in text compression systems, image compression standards such as JPEG, multimedia transmission, fax communication, and file archiving systems. Due to its low computational complexity and efficient coding structure, Huffman Coding remains suitable for real-time applications and embedded communication systems [8]. Although modern compression algorithms have introduced more advanced approaches, Huffman Coding continues to serve as a foundation for many hybrid compression techniques and educational research studies.

This research paper is organized into multiple sections. The literature review discusses previous research contributions related to data compression and Huffman Coding. The methodology section explains the implementation steps of Huffman Encoding and Decoding using MATLAB. Experimental results and performance analysis are then presented, followed by a discussion on

ISSN No : -2456-2165 https://doi.org/10.38124/ijisrt/26may1486

advantages, limitations, applications, and future improvements of the proposed system.

II. PROBLEM STATEMENT

The exponential growth of digital data in modern communication systems, cloud storage platforms, multimedia applications, and internet services has created significant challenges related to efficient data storage and transmission. Large volumes of textual, audio, image, and video data require substantial memory space and high bandwidth for communication. Traditional fixed-length coding techniques allocate the same number of bits to every symbol regardless of their frequency of occurrence, leading to inefficient utilization of storage resources and communication channels.

In many real-time applications, excessive data size increases transmission delay, network congestion, storage cost, and system complexity. Therefore, there is a critical need for an efficient compression technique capable of reducing data size without affecting the integrity and accuracy of the original information. Lossless compression methods are especially important in applications where exact reconstruction of data is required, such as text documents, executable files, medical records, and scientific datasets.

Huffman Coding is a widely used lossless compression technique that addresses these challenges by generating variable-length binary codes based on the statistical probability of symbols. Frequently occurring symbols are assigned shorter binary codes, while less frequent symbols receive longer codes, thereby minimizing the average code length and improving compression efficiency. However, the practical implementation and performance analysis of Huffman Coding require a systematic computational environment for encoding, decoding, and evaluating compression parameters.

This research focuses on the implementation of Huffman Coding and Decoding using MATLAB to develop an efficient lossless data compression system. The study aims to analyze how Huffman Coding reduces redundancy in digital data, improves storage efficiency, and ensures accurate reconstruction of the original information after decompression. The performance of the proposed system is evaluated using parameters such as compression ratio, coding efficiency, and redundancy to determine the effectiveness of the compression process in modern digital communication applications.

III. OBJECTIVES OF THE RESEARCH

The primary objective of this research is to study and implement an efficient lossless data compression technique using Huffman Coding and Decoding in MATLAB. The research focuses on reducing data redundancy and improving storage and transmission efficiency through optimal variable-length coding methods. The specific objectives of the proposed work are as follows:

- To study the fundamental principles of data compression and understand the significance of lossless compression techniques in modern digital communication systems.
- To analyze the working mechanism of Huffman Coding including symbol probability calculation, Huffman tree construction, and variable-length code generation.
- To implement Huffman Encoding and Decoding algorithms in MATLAB for efficient compression and accurate reconstruction of digital data.
- To generate optimal prefix codes based on the frequency of occurrence of input symbols in order to minimize the average code length.
- To evaluate the performance of the proposed compression system using parameters such as compression ratio, coding efficiency, average code length, and redundancy.
- To compare the compressed data size with the original data size and analyze the effectiveness of Huffman Coding in reducing storage requirements.
- To verify the lossless nature of the algorithm by ensuring that the decoded output exactly matches the original input data without any information loss.
- To study the practical applications of Huffman Coding in areas such as text compression, image processing, multimedia communication, and file storage systems.
- To identify the advantages and limitations of Huffman Coding and examine its suitability for modern digital communication and data processing applications.
- To provide a foundation for future research involving adaptive Huffman Coding, hybrid compression methods, and advanced intelligent compression techniques.

IV. LITERATURE REVIEW

Several researchers have contributed significantly to the development and improvement of data compression techniques. Huffman Coding remains one of the most effective methods for statistical lossless compression.

David A. Huffman introduced the Huffman Coding algorithm in 1952 and demonstrated that variable-length prefix coding could minimize average code length based on symbol frequency. His work established the foundation for modern lossless compression methods.

Claude Shannon developed the theory of information entropy, which greatly influenced compression algorithms. Shannon's entropy concept provides the theoretical limit for lossless compression efficiency.

Research by various scholars has shown that Huffman Coding performs efficiently in text compression, image compression, and multimedia applications. MATLAB-based implementations have also been widely adopted in academic and industrial research due to MATLAB's flexible programming environment and built-in functions.

Recent studies have focused on improving Huffman Coding using hybrid methods, adaptive coding, and integration with machine learning techniques. Although newer algorithms such as Arithmetic Coding and Lempel-Ziv-Welch (LZW) provide improved performance in some

ISSN No : -2456-2165 https://doi.org/10.38124/ijisrt/26may1486

scenarios, Huffman Coding remains popular because of its simplicity and lower computational complexity.

V. FUNDAMENTALS OF HUFFMAN CODING

Huffman Coding is a variable-length coding algorithm that assigns shorter binary codes to frequently occurring symbols and longer codes to less frequent symbols.

➢ The algorithm follows these steps:

- Calculate the frequency of each symbol.
- Arrange symbols in ascending order of probability.
- Construct a binary tree by combining the two lowest-frequency nodes repeatedly.
- Assign binary values:

✓ Left branch = 0
✓ Right branch = 1

- Generate unique binary codes for each symbol.
- Encode the input data using generated codes.

The generated codes satisfy the prefix property, meaning no code is the prefix of another code.

VI. HUFFMAN TREE CONSTRUCTION

Huffman Tree Construction is the fundamental process used in Huffman Coding to generate optimal variable-length binary codes for data compression. The Huffman tree is a binary tree structure in which symbols with lower probabilities are placed deeper in the tree, while symbols with higher probabilities are positioned closer to the root node. This arrangement minimizes the average code length and improves compression efficiency.

The construction of the Huffman tree is based on the statistical occurrence of symbols in the input dataset. Symbols that appear more frequently are assigned shorter binary codes, whereas symbols with lower occurrence frequencies receive longer codes. The resulting coding structure satisfies the prefix property, ensuring that no codeword becomes the prefix of another codeword, thereby enabling accurate and unambiguous decoding.

➢ Mathematical Representation of Huffman Coding

Consider a set of source symbols:

$$S = \{s_1, s_2, s_3, \dots, s_n\} \quad (1)$$

where each symbol $s_i$has an associated probability of occurrence P_i.

➢ The probability distribution satisfies the condition:

$$\sum_{i=1}^{n} P_i = 1 \quad (2)$$

➢ The objective of Huffman Coding is to minimize the average code length L_{avg}, which is expressed as:

$$L_{avg} = \sum_{i=1}^{n} P_i \, l_i \quad (3)$$

Where:

- P_i= Probability of occurrence of symbol s_i
- l_i= Length of the binary code assigned to symbol s_i

The Huffman algorithm generates an optimal prefix code such that the average code length becomes as close as possible to the entropy of the source.

The entropy of the source is given by:

$$H = -\sum_{i=1}^{n} P_i \log_2 P_i \quad (4)$$

According to information theory, the average code length satisfies the condition:

$$H \leq L_{avg} < H + 1 \quad (5)$$

This inequality proves that Huffman Coding produces near-optimal compression performance.

➢ Steps for Huffman Tree Construction

The Huffman tree is constructed using the following systematic procedure:

➢ *Step 1: Determine Symbol Frequencies*

The frequency or probability of occurrence of each symbol in the input dataset is calculated.

Consider the following example:

Table 1 Determine Symbol Frequencies

Symbol	Frequency
A	45
B	13
C	12
D	16
E	9
F	5

The total number of symbols is: $45 + 13 + 12 + 16 + 9 + 5 = 100$

ISSN No : -2456-2165 https://doi.org/10.38124/ijisrt/26may1486

The probability of each symbol is calculated as:

$$P_i = \frac{f_i}{\Sigma f_i} \quad (6)$$

Where $f_i$represents the frequency of the symbol.

➢ *Step 2: Arrange Symbols in Ascending Order*

The symbols are arranged according to increasing probability values:

Table 2 Arrange Symbols in Ascending Order

Symbol	Frequency
F	5
E	9
C	12
B	13
D	16
A	45

➢ *Step 3: Combine the Two Lowest-Frequency Nodes*

The two nodes with the smallest frequencies are combined to create a new parent node.

For example:

$$5 + 9 = 14$$

Thus, symbols $F$and $E$are merged into a new node with frequency 14.

The updated list becomes:

Table 3 Combine the Two Lowest-Frequency Nodes

Node	Frequency
C	12
B	13
FE	14
D	16
A	45

➢ *Step 4: Repeat the Combination Process*

The process continues by repeatedly combining the two smallest nodes until only one root node remains.

Next combinations:

$$12 + 13 = 25$$

$$14 + 16 = 30$$

$$25 + 30 = 55$$

$$45 + 55 = 100$$

The final node with frequency 100 becomes the root node of the Huffman tree.

VII. HUFFMAN TREE GENERATION

➢ The generated Huffman tree is represented as a binary tree where:

- Left branch is assigned binary value ‘0’
- Right branch is assigned binary value ‘1’

The binary code for each symbol is obtained by tracing the path from the root node to the corresponding leaf node.

An example of generated Huffman codes is shown below:

Table 4 The generated Huffman tree is represented as a binary tree

Symbol	Huffman Code
A	0
B	101
C	100
D	111
E	1101
F	1100

The symbol with the highest probability (A) receives the shortest code, while symbols with lower probabilities receive longer binary codes.

➢ *Average Code Length Calculation*

The average code length for the generated codes can be calculated using:

ISSN No : -2456-2165 https://doi.org/10.38124/ijisrt/26may1486

$$L_{avg} = \sum P_i l_i \quad (7)$$

Substituting the probabilities and code lengths:

$$L_{avg} = (0.45 \times 1) + (0.13 \times 3) + (0.12 \times 3) + (0.16 \times 3) + (0.09 \times 4) + (0.05 \times 4)$$

$$L_{avg} = 0.45 + 0.39 + 0.36 + 0.48 + 0.36 + 0.20$$

$$L_{avg} = 2.24 \text{ bits/symbol}$$

This value is significantly lower than fixed-length coding, demonstrating the effectiveness of Huffman Coding in reducing storage requirements.

➢ *Advantages of Huffman Tree Construction*

The Huffman tree construction method offers several important advantages:

- Generates optimal prefix codes
- Minimizes average code length
- Improves compression efficiency
- Reduces storage and transmission requirements
- Ensures lossless reconstruction of data
- Provides efficient implementation in real-time communication systems

➢ *Role of Huffman Tree in MATLAB Implementation*

In the proposed work, the Huffman tree is generated in MATLAB using built-in Huffman functions. MATLAB automatically computes symbol probabilities, constructs the Huffman tree, generates binary code dictionaries, and performs encoding and decoding operations efficiently.

VIII. MATLAB IMPLEMENTATION

MATLAB is used for implementing Huffman Encoding and Decoding because of its efficient matrix handling and visualization capabilities.

➢ *Encoding Algorithm*

The encoding process includes:

- Reading input data
- Calculating symbol frequencies
- Generating Huffman dictionary
- Converting symbols into binary codes
- Producing compressed output

```MATLAB
clc;
clear all;
close all;

symbols = ['A' 'B' 'C' 'D' 'E' 'F'];
prob = [0.45 0.13 0.12 0.16 0.09 0.05];

dict = huffmandict(symbols, prob);

disp('Huffman Dictionary:');
disp(dict);

inputSig = ['A' 'B' 'A' 'C' 'D' 'A'];

encodedSig = huffmanenco(inputSig, dict);

disp('Encoded Signal:');
disp(encodedSig);
```

Fig 1 MATLAB Code for Huffman Encoding

➢ *Decoding Algorithm*

The decoding process reconstructs the original data from compressed binary codes.

ISSN No : -2456-2165 https://doi.org/10.38124/ijisrt/26may1486

```
MATLAB

decodedSig = huffmandeco(encodedSig, dict);

disp('Decoded Signal:');
disp(decodedSig);
```

Fig 2 MATLAB Code for Huffman Decoding

IX. FLOWCHART OF PROPOSED SYSTEM

➢ *Encoding Process*

- Start
- Input Data
- Calculate Symbol Frequency
- Generate Huffman Tree
- Assign Binary Codes
- Encode Data
- Display Compressed Output
- Stop

➢ *Decoding Process*

- Start
- Read Encoded Data
- Traverse Huffman Tree
- Decode Symbols
- Reconstruct Original Data
- Stop

X. PERFORMANCE METRICS

The efficiency of Huffman Coding is evaluated using the following metrics:

➢ *Compression Ratio*

Compression Ratio is one of the most important parameters used to measure the effectiveness of a compression technique. It represents the ratio between the size of the original data and the size of the compressed data. A higher compression ratio indicates better compression performance and more efficient utilization of storage space and transmission bandwidth.

The compression ratio is mathematically expressed as:

$$Compression\ Ratio = \frac{Size\ of\ Original\ Data}{Size\ of\ Compressed\ Data} \quad (8)$$

If the compressed data size becomes significantly smaller than the original data size, the compression algorithm is considered efficient. In Huffman Coding, the compression ratio depends on the probability distribution of the symbols in the input dataset.

➢ *Average Code Length*

Average code length refers to the average number of bits required to represent each symbol after Huffman Encoding. Since Huffman Coding assigns variable-length binary codes to symbols, the average code length is generally smaller than fixed-length coding methods.

The average code length is calculated using the weighted sum of symbol probabilities and their corresponding code lengths:

$$L_{avg} = \sum_{i=1}^{n} P_i \times l_i \quad (9)$$

Where:

- P_i= Probability of occurrence of the i^{th}symbol
- l_i= Length of the Huffman code assigned to the i^{th}symbol

A lower average code length indicates better coding efficiency and improved compression performance.

➢ *Coding Efficiency*

Coding efficiency measures how effectively the Huffman Coding algorithm utilizes the available bits in comparison to the theoretical entropy limit defined by information theory. Higher efficiency indicates that the generated Huffman codes are close to the optimal coding structure.

The coding efficiency is calculated as:

$$Efficiency = \frac{Entropy}{Average\ Code\ Length} \times 100 \quad (10)$$

Coding efficiency is usually expressed as a percentage. A value closer to 100% indicates that the compression technique performs near the theoretical optimum.

➢ *Entropy*

Entropy represents the average amount of information contained in a message source. It defines the theoretical minimum number of bits required to represent the data without losing information. Entropy is calculated using the probability distribution of symbols.

The entropy equation is given by:

$$H = -\sum_{i=1}^{n} P_i \log_2 P_i \quad (11)$$

ISSN No : -2456-2165 https://doi.org/10.38124/ijisrt/26may1486

Where:

- H= Entropy of the source
- P_i= Probability of occurrence of the i^{th}symbol

Entropy provides a benchmark for evaluating the effectiveness of Huffman Coding.

➢ *Redundancy*

Redundancy represents the difference between the actual average code length and the theoretical entropy value. It indicates the extra bits used during the encoding process beyond the theoretical minimum requirement.

The redundancy is expressed as:

$$Redundancy = 1 - Efficiency \quad (12)$$

Lower redundancy indicates a more efficient compression system.

➢ *Reconstruction Accuracy*

Reconstruction accuracy measures the capability of the decoding process to recover the original data exactly from the compressed bit stream. Since Huffman Coding is a lossless compression technique, the decoded output should be identical to the original input data without any information loss.

Reconstruction accuracy can be verified by comparing the original input sequence with the decoded output sequence generated after Huffman Decoding.

XI. EXECUTION TIME

Execution time represents the total time required for encoding and decoding operations in MATLAB. It provides information regarding the computational complexity and real-time applicability of the compression system.

Efficient execution time is important in practical communication systems where fast compression and decompression are required for real-time data transmission.

➢ *Memory Utilization*

Memory utilization refers to the amount of memory required for storing the Huffman tree, code dictionary, encoded data, and decoding structures during implementation. Efficient memory utilization improves the overall system performance and reduces hardware requirements.

XII. RESULTS AND DISCUSSION

The MATLAB simulation demonstrates that Huffman Coding effectively compresses textual data by reducing average code length. Symbols with higher occurrence frequencies receive shorter codes, leading to reduced storage requirements.

The decoded output perfectly matches the original input sequence, confirming the lossless nature of the algorithm. The implementation also shows that Huffman Coding achieves significant compression for datasets with non-uniform symbol distributions.

➢ *Observations*

- Compression efficiency increases when symbol probability distribution is uneven.
- The algorithm is simple and computationally efficient.
- MATLAB implementation provides accurate encoding and decoding operations.
- Huffman Coding is suitable for text, image, and multimedia compression applications.

XIII. ADVANTAGES OF HUFFMAN CODING

- ➢ Lossless compression technique
- ➢ Efficient storage utilization
- ➢ Reduced transmission bandwidth
- ➢ Simple implementation
- ➢ Optimal prefix coding
- ➢ Accurate reconstruction of original data

XIV. LIMITATIONS OF HUFFMAN CODING

- ➢ Less effective for uniformly distributed data
- ➢ Requires frequency table generation
- ➢ Static Huffman Coding may not adapt well to changing data patterns
- ➢ Large datasets may increase tree construction complexity

XV. APPLICATIONS

➢ Huffman Coding is widely used in:

- Text file compression
- Image compression
- JPEG compression
- Multimedia communication
- Fax transmission
- File archiving systems
- Wireless communication systems

XVI. FUTURE SCOPE

➢ Future improvements may include:

- Adaptive Huffman Coding implementation
- Hybrid compression techniques
- Integration with Artificial Intelligence algorithms
- Real-time hardware implementation
- Enhanced multimedia compression systems

XVII. CONCLUSION

This research paper presented the implementation of Huffman Coding and Decoding using MATLAB for efficient lossless data compression. The proposed approach

ISSN No : -2456-2165 https://doi.org/10.38124/ijisrt/26may1486

successfully reduced data size while preserving complete information accuracy. MATLAB simulations demonstrated the effectiveness of variable-length coding in minimizing average code length and improving storage efficiency.

The study confirmed that Huffman Coding remains a reliable and computationally efficient compression technique for digital communication and multimedia applications. Although modern compression methods provide additional enhancements, Huffman Coding continues to serve as a fundamental and widely adopted algorithm in the field of data compression.

REFERENCES

[1]. Introduction to Data Compression, 5th ed. Burlington, MA, USA: Morgan Kaufmann, 2017.
[2]. Elements of Information Theory, 2nd ed. Hoboken, NJ, USA: Wiley-Interscience, 2006.
[3]. David A. Huffman, “A Method for the Construction of Minimum-Redundancy Codes,” *Proceedings of the IRE*, vol. 40, no. 9, pp. 1098–1101, Sep. 1952.
[4]. Claude Shannon, “A Mathematical Theory of Communication,” *Bell System Technical Journal*, vol. 27, pp. 379–423, Jul. 1948.
[5]. Data Compression: The Complete Reference, 4th ed. London, U.K.: Springer, 2007.
[6]. MATLAB Documentation, MathWorks Official Website
[7]. Digital Communications, 5th ed. New York, NY, USA: McGraw-Hill, 2007.
[8]. Information Theory and modern lossless compression research studies.