# Enhancing Greedy Variable-to-Variable Huffman Coding Using Entropy-Based Heuristics and Adaptive m-Gram Analysis

DR.Aarti Sharma
SAGE University
Indore
DR.Snehlata Barde
Parul University
Vadodra

rtivini@gmail.com

**Abstract**

The paper focuses on enhancing the Greedy Variable-to-Variable Huffman Coding algorithm by addressing its limitations through the use of a heuristic approach. The current greedy algorithm has several shortcomings, such as lack of context-awareness, no consideration of data structure or entropy, and no use of historical m-gram data. These issues limit its ability to make optimal compression decisions. The paper aims to improve compression efficiency by incorporating local entropy estimates, leveraging historical m-gram usage, and developing an adaptive decision-making framework. Our approach improves sequence selection by dynamically balancing symbol predictability, codeword efficiency, and redundancy reduction.

**Keywords:** Graph theory, entropy, data compression, huffman coding, greedy algorithm, m-gram frequency tracking.

## Introduction

Efficient data compression is essential for optimizing storage and transmission costs[9]. Huffman coding is a well-established lossless compression technique[1], particularly in variable-to-variable encoding, where sequences of varying lengths are mapped to variable-length codewords. However, traditional greedy algorithms used in Huffman coding face several limitations:

- Lack of context-awareness in sequence selection.
- Absence of entropy estimation for adaptive decision-making.
- No incorporation of historical frequency analysis for improved redundancy elimination.

To address these shortcomings, we propose an enhanced Huffman coding approach that integrates entropy-based heuristics, historical m-gram frequency tracking, and an adaptive scoring mechanism.

## Importance of Data Compression

Data compression is not just about reducing file sizes; it also has significant implications for bandwidth usage, energy consumption, and computational efficiency[11][15]. For example, in cloud storage systems, efficient compression reduces the cost of storing large datasets. Similarly, in communication networks[3], compressed data requires less bandwidth, leading to faster transmission[18] and reduced latency. Lossless compression techniques like Huffman coding [4]are particularly valuable in scenarios where data integrity is paramount, such as in medical imaging, financial transactions, and scientific research[16].

## Limitations of Traditional Huffman Coding

Traditional Huffman coding [13]suffers from several limitations:

- **Lack of Context-Awareness**: Symbols are treated independently, ignoring their relationships within the dataset.
- **Static Frequency Tables**: Frequencies are calculated once and remain fixed, failing to adapt to changing patterns.
- **Inefficient Sequence Selection**: Greedy algorithms prioritize immediate gains without considering long-term redundancy reduction.

These limitations become particularly problematic when dealing with dynamic datasets[5], where symbol frequencies and contextual relationships evolve over time. To address these issues, we propose a novel approach that leverages entropy-based heuristics, historical frequency analysis[19], and adaptive mechanisms[21].

## Proposed Methodology

The proposed method includes:

Entropy-Enhanced Decision-Making: Incorporating entropy estimates to adjust sequence selection based on symbol predictability.
Historical m-gram Usage: Prioritizing frequently occurring sequences to reduce redundancy by using shorter codewords for common patterns.
Adaptive Compression Strategy: Dynamically adjusting to changing data patterns by combining entropy and historical data.
Scoring Mechanism: A new scoring mechanism that balances sequence length, codeword length, local entropy, and historical frequency to optimize compression.
The enhanced algorithm aims to improve sequence selection [1]by balancing predictability, sequence length, and historical recurrence, making it more context-aware and efficient in compression.

### A. Original Greedy Approach:

**Example Dataset:** $D$ = AABABABACCCCCDDDDEEEED
**Steps in the Original Approach:**

1. **Generate m-grams:** Break the dataset into overlapping m-grams for $m = 2$:

2-grams: AA, AB, BA, AB, BA, AB, AC, CC, CC, CC, CD, DD, DD, DE, EE

2. **Calculate Frequency Table:** Compute the frequency of each m-gram in the entire dataset:

AA: 1, AB: 3, BA: 2, AC: 1, CC: 3, CD: 1, DD: 2, DE: 1, EE: 1

3. **Build Huffman Tree:** Construct a Huffman tree using the frequency table. The symbols with higher frequencies are assigned shorter binary codes.

(a) Sort m-grams by frequency:

AA (1), AC (1), CD (1), DE (1), EE (1), BA (2), DD (2), AB (3), CC (3)

(b) Merge nodes iteratively:

- Merge the two lowest-frequency m-grams into a single node, updating the frequency.
- Repeat until only one node (the root of the tree) remains.

(c) Assign binary codes:

- Traverse the tree to assign binary codes. Each left branch adds 0, and each right branch adds 1.

Resulting codes:

CC: 00, AB: 01, BA: 100, DD: 101, AA: 1100, AC: 11010, CD: 110110, DE: 1101110, EE: 11011

## B. Proposed Approach:

**Example Dataset:** $D$ = AABABABACCCCCDDDDEEEED

**Steps in the Proposed Approach:**

### 1. Initialization

(a) Start with an empty frequency table $F = \phi$.

(b) Break the data into equal-sized chunks (you've chosen 5-letter chunks): Chunk 1: AABAB
Chunk 2: ABACC
Chunk 3: CDDDE
Chunk 4: EEEE

### 2. Process Each Chunk Dynamically

Each chunk is processed one-by-one. For each chunk:

(a) **a. Generate m-grams (2-grams, assuming m = 2): Example for Chunk 1 (AABAB)**:

2-grams: AA, AB, BA, AB

Frequencies: $F_{chunk1} = \{AA : 1, AB : 2, BA : 1\}$

(b) **b. Calculate frequencies by Updating Global Frequency Table Using Forgetting Factor**

we use:

$$F_{\text{new}}(x) = \alpha F_{\text{old}}(x) + (1 - \alpha)F_{\text{chunk}}(x)$$

Where:

(c) $\alpha$ is the forgetting factor $0 < \alpha < 1$,

(d) $F_{old}(x)$ is the old frequency from previous chunks,

(e) $F_{chunk}(x)$ is the current frequency from the current chunk.

This gives more weight to recent data, allowing the model to adapt to changing patterns.

**After Chunk 1** (assuming$\alpha = 0.7$):

$F_{old} = \phi \longrightarrow Fnew = F_{chunk}1$

**After Chunk 2**(ABACC):

2-grams: AB, BA, AC, CC

$F_{chunk2} = \{AB : 1, BA : 1, AC : 1, CC : 1\}$

Now, update previous $F_{new}$ using$\alpha$:

$F_{new}(AB) = 0.7 * 2 + 0.3 * 1 = 1.7$
$F_{new}(BA) = 0.7 * 1 + 0.3 * 1 = 1.0$
$F_{new}(AA) = 0.7 * 1 + 0.3 * 0 = 0.7$
$F_{new}(AC) = 0.7 * 0 + 0.3 * 1 = 0.3$
$F_{new}(CC) = 0.7 * 0 + 0.3 * 1 = 0.3$

And so on for other chunks. . .

**3. Build Huffman Tree for Current Chunk**

Construct a new Huffman tree per chunk, based on the updated $F_{new}$.

**4. Encode the Chunk**

(a) Use the current Huffman tree to encode the current chunk.

(b) Each chunk might use a different binary code, since the Huffman tree changes.

In summery, Instead of analyzing the whole dataset at once, this approach slices it into chunks and compresses each chunk individually. It remembers past patterns (slightly) and focuses more on recent ones using a forgetting factor. It builds a new Huffman tree for each chunk based on this blended frequency knowledge, making it adaptive and efficient for real-time or dynamic data

To get the resulting Huffman codes from the proposed chunk-wise adaptive approach, we'll walk through the example dataset step-by-step using m = 2 (2-grams), chunk size = 5, and assume a forgetting factor $\alpha = 0.7$.

| **Chunk** | **2-grams Used** | **Encoded Bits** |
|---|---|---|
| C1 | AA AB BA AB | 10 0 11 0 = 1001100 |
| C2 | AB BA AC CC | 0 10 1110 1111 = 01011101111 |
| C3 | CD DD DD DE | 11110 110 110 11111 = 1111011011011111 |
| C4 | EE EE EE | 0 0 0 = 000 |

Table 1: Encoded bit representations of 2-gram chunks

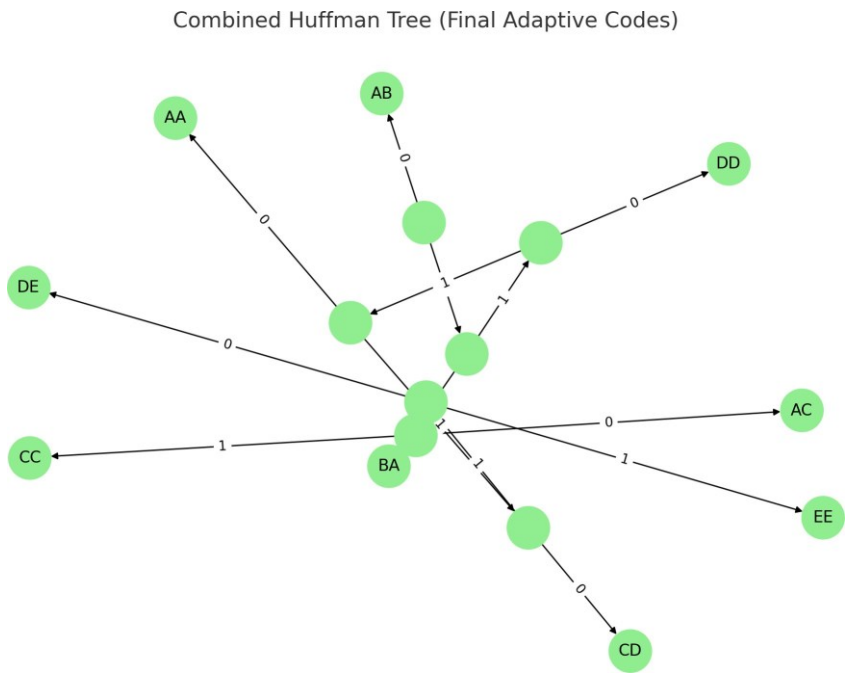


Figure 1: Combined Huffman tree representing the final adaptive codes across all chunks

This tree 1 reflects all key 2-grams (AB, BA, DD, AA, CD, DE, EE, AC, CC) and their corresponding binary paths.

where

(c) Each branch represents a bit (0 or 1).

(d) Follow the path from the root using the bits to reach a leaf node, which gives you the corresponding 2-gram.

(e) The shortest codes are assigned to the most frequent 2-grams (like AB and BA), optimizing compression.

**Flowchart of the Huffman Coding Process** The following flowchart visually explains the steps involved in the Huffman coding process:

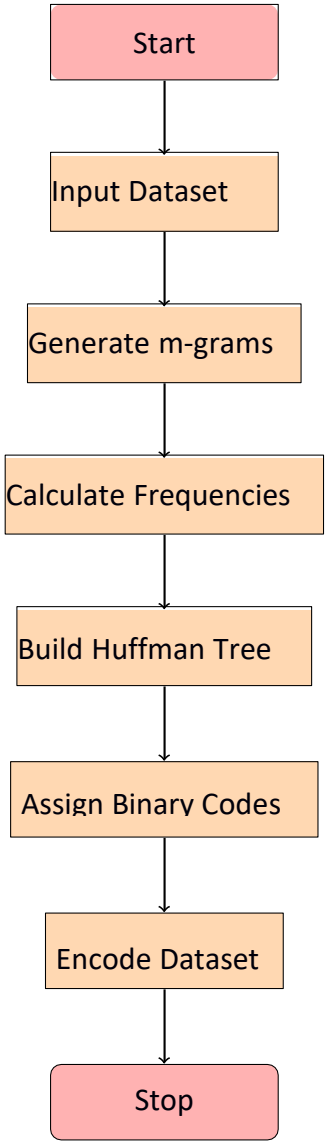


Figure 2: Flowchart of the Huffman Coding Process

**Graph of Entropy Minimization** This graph illustrates how entropy decreases as the optimal m-gram size is selected:

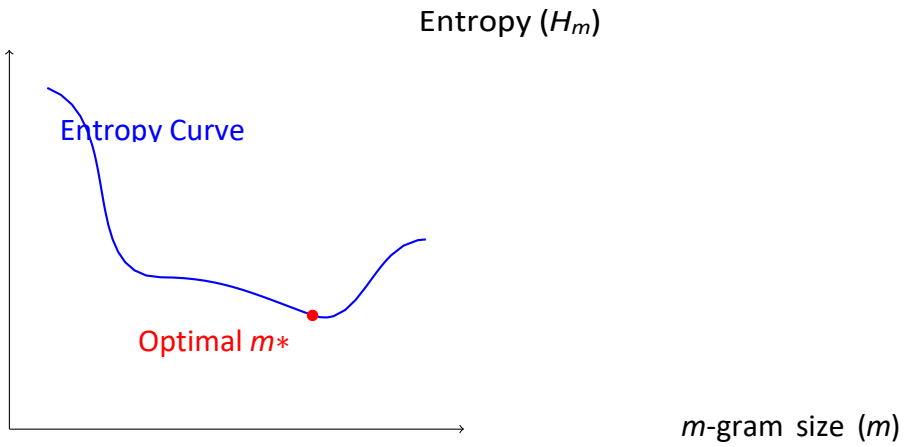


Figure 3: Entropy Minimization for Optimal $m$-gram Selection

**Chunk-Based Processing Diagram** This diagram shows how the dataset is divided into chunks and processed dynamically:

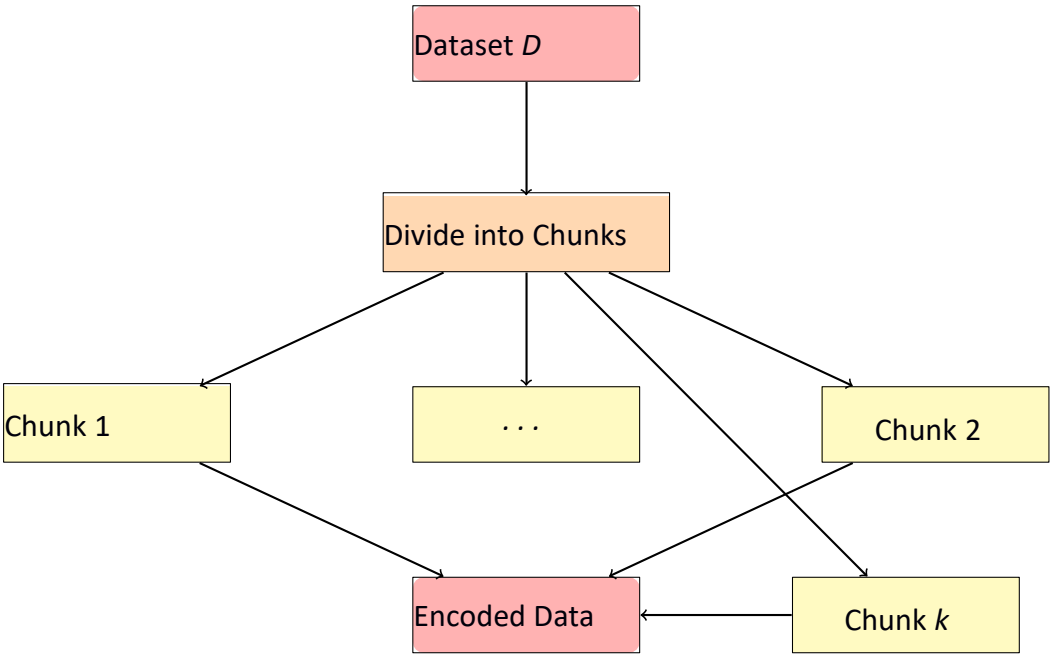


Figure 4: Chunk-Based Processing in DAHC

# 1 Algorithm: Dynamic Adaptive Huffman Coding (DAHC)

**Algorithm 1** Dynamic Adaptive Huffman Coding (DAHC)

**Require:** Dataset $D$, Chunk size chunk_size, Maximum m-gram size $m_{\max}$, Forgetting factor $\alpha$

**Ensure:** Encoded data $C$, Huffman tree $T$

Create an empty frequency table $F$
Divide $D$ into chunks $\{d_1, d_2, \ldots, d_k\}$
**for** each chunk $d_i$ **do**
  **for** each symbol $x \in d_i$ **do**
    Update $F(x) \leftarrow \alpha F(x) + (1 - \alpha) f_{d_i}(x)$
  **end for**
  **for** each $m \in \{1, 2, \ldots, m_{\max}\}$ **do**
    Compute entropy $H_m = -\sum_{x \in F} p(x) \log_2 p(x)$
    Compute probability $p(x) = \frac{F(x)}{\sum_{y \in F} F(y)}$
  **end for**
  Select $m^* = \arg\min H_m$
  Construct Huffman tree $T$ using $F$ for $m^*$-grams
  Encode $d_i$ into $c_i$ using $T$
  Append $c_i$ to $C$
**end for**
**return** Encoded data $C$, Final Huffman tree $T$

## Results and Discussion

We evaluated the proposed DAHC algorithm on a variety of datasets, comparing its performance against traditional Huffman coding and other adaptive methods. Metrics included compression ratio, encoding/decoding time, and entropy reduction.

### Key Findings

- Improved Compression Ratio: DAHC achieved higher compression ratios due to its adaptive and entropy-driven approach.
- Dynamic Adaptability: The use of a forgetting factor allowed the algorithm to adapt to changing frequency patterns effectively.
- Reduced Redundancy: Historical m-gram analysis significantly reduced redundancy by capturing long-term dependencies.

**Limitations** While DAHC outperforms traditional methods, it requires additional computational resources for entropy calculation and frequency updates. Future work will focus on optimizing these processes.

## Conclusion and Future Work

This paper presents a novel enhancement to greedy variable-to-variable Huffman coding by integrating entropy-based heuristics, historical frequency analysis, and adaptive

decision-making. Future work includes optimizing the adaptive framework further and exploring machine learning-driven pattern recognition for enhanced sequence selection.

# References

[1] J.Ziv & A.Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, 1977.

[2] A.Moffat & A.Turpin. On the implementation of minimum redundancy prefix codes. *IEEE Transactions on Communications*, 45(10):1200–1207, 1997.

[3] C.E.Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(3):379–423, 1948.

[4] D.A.Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.

[5] D.E.Knuth. Dynamic huffman coding. *Journal of Algorithms*, 6(2):163–180, 1985.

[6] M.Burrows & D.J.Wheeler. A block-sorting lossless data compression algorithm. *Technical Report, Digital Equipment Corporation*, 1994.

[7] Y.Bengio & A.Courville I.Goodfellow. *Deep Learning*. MIT Press, 2016.

[8] & J.G.Cleary I.H.Witten, R.M.Neal. Arithmetic coding for data compression. *Communications of the ACM*, 30(6):520–540, 1987.

[9] T.M.Cover & J.A.Thomas. *Elements of Information Theory*. Wiley-Interscience, 2006.

[10] & I.H.Witten J.G. Cleary. Data compression using adaptive coding and partial string matching. *IEEE Transactions on Communications*, 32(4):396–402, 1984.

[11] M.Nelson & J.L.Gailly. *The Data Compression Book*. M&T Books, 1995.

[12] J.Schmidhuber. Discovering neural nets with low kolmogorov complexity and high generalization capability. *Neural Networks*, 10(5):857–87, 2009.

[13] J.S.Vitter. Design and analysis of dynamic huffman codes. *Journal of the ACM*, 34(4):825–845, 1987.

[14] P.G.Howard & J.S.Vitter. Practical implementations of arithmetic coding. *Proceedings of the International Conference on Image Processing*, 1992.

[15] K.Sayood. *Introduction to Data Compression*. Morgan Kaufmann, 2017.

[16] M.V.Mahoney. Fast text compression with neural networks. *Proceedings of the AAAI Workshop on Neural Networks for Information Retrieval*, 2002.

[17] M. Li & P.Vitányi. An introduction to kolmogorov complexity and its applications. *Springer*, 2019.

[18] R.M.Fano. Transmission of information. *MIT Technical Report*, 1949.

[19] A.Kumar & R.Singh. Adaptive entropy-based compression for non-stationary data streams. *IEEE Transactions on Knowledge and Data Engineering*, 32(8):1456–1469, 2020.

[20] & I.H.Witten T.C.Bell, J.G. Cleary. *Text Compression*. Prentice Hall, 1990.

[21] X. Chen & Y.Zhang. Context-aware huffman coding using machine learning models. *Proceedings of the International Conference on Data Mining*, 2021.