Adaptive Huffman Algorithm with Text Clustering and Multiple Character Modification for Data Compression

1st Harshul Jain
Department of Information Technology
National Institute of Technology, Karnataka
Signals and Systems
231IT025

2nd Nikhil Agarwal
Department of Information Technology
National Institute of Technology, Karnataka
Signals and Systems
231IT044

***Abstract*—This project presents an enhanced data compression technique using the Adaptive Huffman Algorithm, combined with text clustering and multiple character modification, to improve compression efficiency for textual data. By grouping similar sequences and reducing redundancy, the modified algorithm achieves superior compression ratios compared to traditional methods. Experimental results indicate a significant improvement in compression ratio without substantial compromise in execution time, making this approach viable for diverse datasets.**

***Index Terms*—Adaptive Huffman, data compression, text clustering, multiple character modification, algorithm efficiency**

I. Introduction

Data compression techniques aim to reduce storage size without compromising data integrity. Adaptive Huffman coding, a popular lossless compression algorithm, dynamically adjusts based on the input data stream, allowing high compression ratios. However, it encounters limitations with repetitive or similar patterns in textual data. This study introduces text clustering and multiple character modification to the Adaptive Huffman Algorithm to address these limitations. By identifying and clustering repetitive patterns, the algorithm compresses data more effectively, achieving a better compression ratio than conventional techniques.

II. Literature Survey

Data compression has evolved with various algorithms such as Huffman coding, Run-Length Encoding, and Lempel-Ziv-Welch (LZW). Adaptive Huffman coding extends traditional Huffman coding by dynamically updating the codebook. The methodolgy used has incorporated clustering techniques and multiple character encoding to improve compression, . Multiple symbol encoding further optimizes the compression ratio by encoding frequent character groups together, reducing redundancy.

III. Methodology

This study's methodology involves modifying the Adaptive Huffman algorithm by integrating:

- **Text Clustering**: Identification of frequent clusters within the encoded data using pattern recognition techniques.
- **Character Modification**: Replacing identified clusters with new characters, adjusting the Huffman tree accordingly.

IV. Objective 1

- **Proposed Methodology**: Assess the impact of integrating text clustering and multiple character modification techniques on the compression ratio of the Adaptive Huffman algorithm when applied to structured datasets
- **Frequency Calculation**: Compute how often each character occurs in the input.
- **Huffman Tree**: Build a tree where characters with lower frequency appear deeper in the tree, resulting in shorter binary codes for frequent characters.
- **Pattern Detection**: Detect frequent patterns in the encoded text, replace them with shorter symbols, and reapply Huffman coding to further reduce size

V. Objective 2

- **Proposed Methodology**: Measure and compare the speed of the modified Adaptive Huffman algorithm against the original. Analyze how the clustering and modification techniques affect processing time and evaluate any trade-offs between compression efficiency and speed.
- **Timing function**: To measure and compare the speed of the modified Adaptive Huffman algorithm against the original, we use timing functions to capture the time it takes to encode and compress the text with each approach.

The algorithm begins with traditional Huffman encoding, then applies clustering to identify and replace frequently occurring bit sequences. This process iterates, re-encoding the text until no further compression gains are observed. The modified code is implemented in C++ and uses pattern frequency analysis for clustering.

VI. Setup

The experiment setup includes:

- **Environment**: The algorithm was developed and tested using C++ on a standard computing setup.

- **Tools**: C++ for algorithm development and various libraries for data processing and analysis.
- **Dataset**: Standard benchmark text datasets, including common phrases and structured documents, were used to test compression efficiency across varied text formats.
- **Performance Metrics**: Compression ratio, execution time, and memory usage were recorded to evaluate the algorithm's efficiency.

VII. Results and Analysis

The modified algorithm was tested against standard datasets, comparing its performance to traditional Huffman and LZW algorithms. The following metrics were recorded:

- **Compression Ratio**: The modified algorithm achieved an average compression ratio outperming the standard Huffmann compression by 1.36 times
- **Execution Time**: Due to additional clustering and character modification steps, the execution time increased slightly but remained within acceptable limits for large datasets.
- **Memory Usage**: While the clustering process required additional memory, the overall increase was manageable, demonstrating scalability for large datasets.

VIII. Conclusion and Future Work

The enhanced Adaptive Huffman Algorithm with text clustering and multiple character modification effectively compresses textual data, especially those with repetitive patterns. Future research could explore dynamic clustering thresholds and real-time optimization techniques for further efficiency. Additionally, the algorithm could be extended to non-textual data, adapting clustering and modification for various data types.