Information Theory – Final Project

Adam Zelzer ID 328489166, Nitzan Ron ID 215451709

This document provides an overview of the compression algorithm used, detailing the methods applied, the results obtained across various files, and a performance analysis based on compression ratio and processing time.

Methods

In our Algorithm, we used 3 main methods commonly employed in compression algorithms like **zip**, **gzip** and **bzip2**.

Burrows-Wheeler Transform (BWT)

The BWT is an invertible transform, which rearranges the characters of the input data into runs of similar characters, using a suffix array, making it easier to compress using subsequent stages.

To generate the suffix array, we implemented the **SA-IS** algorithm, which runs in linear time $O(n)$, and improved time performance substantially.

Move-to-Front (MTF)

The MTF encoding step takes advantage of the repeated characters generated by the BWT to convert them into small integers, representing the positions of symbols in a dynamically updated list.

In many cases, the output array gives frequently repeated characters' lower indexes which is useful in data compression algorithms, such as LZW.

Lempel-Ziv-Welch (LZW)

The final stage of compression uses LZW, which is a dictionary-based algorithm that replaces repeated sequences of data with shorter codes, thus reducing the overall file size.

Our implementation generated a string of bits as an output, which is then saved in a binary format (.bin).

Why this order?

The compression pipeline uses Burrows-Wheeler Transform (BWT) followed by Move-to-Front (MTF) and Lempel-Ziv-Welch (LZW) for optimal results. The BWT rearranges characters into runs of similar characters, making the data more amenable to compression. The MTF step then leverages the locality of repeated characters by encoding the position of characters in a dynamic dictionary, reducing redundancy in the *data. Finally, LZW is applied, which is a dictionary-based entropy compression technique that efficiently encodes sequences of characters. This particular order ensures that the BWT creates patterns that MTF can better encode, and LZW compresses these repeated patterns further, maximizing compression efficiency.*

Results

The algorithm was tested on five text files of varying sizes. Below are the key performance metrics observed for each file:

File Name	*Or. size (bytes)*	*Comp. size (bytes)*	*Comp. Ratio (%)*	*Comp. time (sec)*	*Decomp. time (sec)*
alice.txt	152,086	51,366	34	2.61	0.68
paradiselost.txt	481,859	171,074	36	10.98	2.64
shakespeare.txt	1,115,394	372,011	33	20.79	3.85
bible.txt	4,047,392	909,651	22	72.76	13.25
dickens.txt	11,046,828	2,986,677	27	134.73	47.32

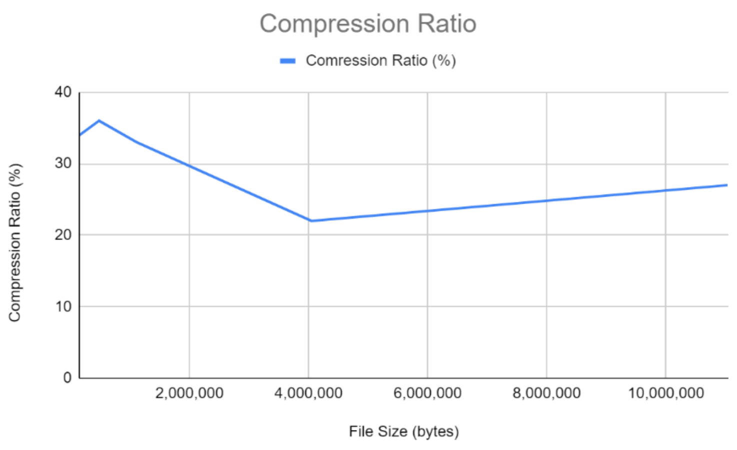
Analysis

Compression Ratio

The compression ratio is calculated as:

$$Compression\ Ratio = \frac{Compressed\ size}{Original\ size} \cdot 100\%$$

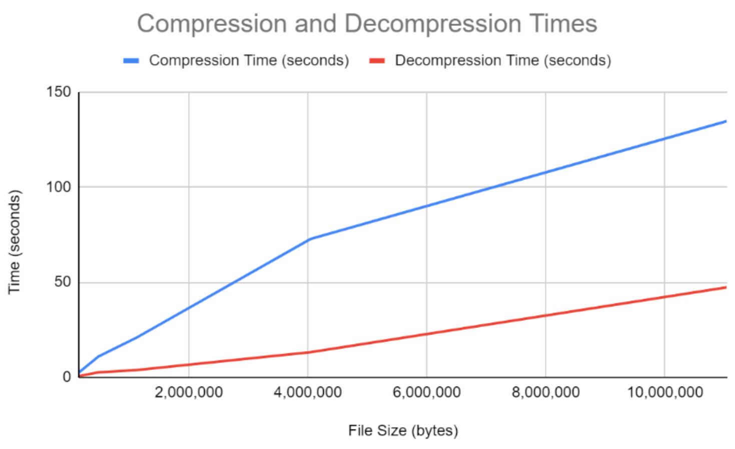
The best compression ratio was achieved with the *bible.txt* file (4,047,392 bytes), which was reduced to 22% of its original size. On average, the algorithm reduced files to 30.4% of their original size.



Compression and Decompression Time

As expected, larger files took more time for both compression and decompression. For example, *dickens* file (11,046,828 bytes), the largest file, took around 135 seconds to compress and 47 seconds to decompress.

On the other hand, smaller files like *alice.txt* file (152,086 bytes), took significantly less time, requiring only 2.61 seconds for compression and 0.68 seconds for decompression.



Conclusion

The combination of Burrows-Wheeler, Move-to-Front, and LZW resulted in efficient compression, especially for large text files, with compression ratio of 30% on average. Although larger files took longer to process, the algorithm demonstrated reasonable performance, achieving a substantial reduction in file size while maintaining manageable compression and decompression times. Additionally, the compression and decompression times seem linear with respect to file size, which is the best we can hope for, as to compress of file we must read it, which is an $O(n)$ operation.

This algorithm shows promise for text-based compression tasks, especially when storage space is at a premium and compression time is less critical.

Github Repository

https://github.com/AdamZlr/Information-Theory