Burrows–Wheeler Huffman Text File Compression

Bernhardt Steyn
Stellenbosch University
Stellenbosch, South Africa
21740178@sun.ac.za

ABSTRACT

This project implements a lossless text compression tool using the Burrows–Wheeler Transform (BWT) followed by Huffman coding. The tool is written in C, supports both compression and decompression, and is tested on thirty synthetic text files. While effective on low-entropy data, the tool is outperformed by bzip2 and xz in terms of compression ratio. Statistical testing using the Friedman test confirms these differences are significant. The implementation is functionally correct and demonstrates the viability of BWT-enhanced Huffman coding, though further improvements are needed to match modern compression utilities.

ACM Reference Format:
Bernhardt Steyn. 2025. Burrows–Wheeler Huffman Text File Compression. In . ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Data compression is a critical component of modern information systems, enabling efficient storage and transmission of large volumes of textual and binary data. Lossless compression algorithms play a central role in this context, particularly when data integrity must be preserved. Among these, Huffman coding [2] remains one of the most widely used entropy coding techniques due to its simplicity and optimality for symbol-wise encoding based on frequency distributions. However, the effectiveness of Huffman coding is inherently tied to the distribution of the input symbols, limiting its compression efficiency on unstructured or low-redundancy input. To address this limitation, the Burrows–Wheeler Transform (BWT) [1] is often employed as a preprocessing step. The BWT is a reversible transformation that reorganizes the input string into a form that exposes redundancies by grouping similar characters together. This transformed output is more amenable to entropy-based encoding techniques, thereby enhancing overall compression performance. The combination of BWT followed by Huffman encoding leverage the strengths of both methods—structural reordering and statistical compression—to achieve improved compression ratios on natural language and structured text files.

This project presents the design and implementation of a command-line tool for lossless compression and decompression of plain text files using the BWT and Huffman coding pipeline. The tool is implemented in C and structured to support modular integration as a standalone library.

The remainder of this document is structured as follows: Section II provides the necessary background on Huffman coding and the Burrows–Wheeler Transform. Section III outlines the implementation details, including file handling, memory usage, and encoding work flows. Section IV describes the evaluation procedure and test methodology. Section V presents the results of empirical testing across a variety of datasets, and Section VI concludes with a summary of the findings and possible directions for future improvement.

2 BACKGROUND

This section provides a background for the relevant information for this paper.

2.1 Data Compression in General

Data compression refers to the process of encoding information using fewer bits than the original representation [5], thereby reducing the size of the data for storage or transmission. It is broadly categorised into two types: lossless and lossy compression. Lossless compression preserves all original data and ensures exact reconstruction, making it suitable for textual, executable, and certain scientific data. In contrast, lossy compression discards less perceptible information to achieve higher compression ratios, and is typically used for multimedia formats such as images, audio, and video.

The fundamental objective of compression is to eliminate statistical redundancy by exploiting patterns or predictability in the data. This is often achieved through a combination of modelling and encoding. Modelling involves analysing the structure or frequency of elements within the data, while encoding transforms the model's predictions into efficient binary representations. Effective compression algorithms aim to strike a balance between compression ratio, computational efficiency, and ease of decompression.

2.2 Huffman Coding

Huffman coding [2] is a variable-length prefix encoding method introduced by David A. Huffman in 1952. It assigns shorter binary codes to more frequent symbols and longer codes to less frequent ones, thereby achieving efficient representation for statistically skewed data. The method constructs a binary tree, known as a Huffman tree, based on the frequency distribution of symbols in the input. This tree is then traversed to assign unique binary codes to each symbol such that no code is a prefix of another. Huffman coding guarantees optimality among all prefix codes for a given symbol distribution, making it highly effective for natural language text and other non-uniform data.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2.3 Burrows–Wheeler Transform

The Burrows–Wheeler Transform [1], introduced in 1994, is a reversible text transformation that enhances the locality of similar characters. Although it does not compress data directly, the BWT permutes the input string into a form that is more amenable to entropy encoding techniques such as run-length encoding or Huffman coding. By sorting all cyclic rotations of the input string and extracting the last column of the resulting matrix, the BWT effectively groups similar characters together. This clustering of repeated symbols significantly improves the compression ratio when followed by traditional coding algorithms.

2.4 Combined Use

The combination of BWT and Huffman coding leverage both structural and statistical properties of the input data [5]. While BWT improves character locality and reveals hidden redundancies, Huffman coding capitalises on symbol frequency to reduce bit-level representation. This pipeline has been adopted in several practical compression tools and is particularly effective for compressing natural language text, where character repetition and uneven frequency distributions are common. The synergy between the two methods forms a robust foundation for the development of high-performance compression utilities.

3 EMPIRICAL PROCEDURE

The primary objective of this evaluation is to assess the performance and correctness of the implemented compression tool that combines the Burrows–Wheeler Transform (BWT) with Huffman coding. This evaluation aims to determine the tool's effectiveness in compressing various types of text input while ensuring accurate reconstruction of the original data. In addition to stand-alone performance, the tool is also compared against two well-established compression utilities, `bzip2`[6] and `xz` [4], to provide context for its practical viability.

3.1 Test Data

Thirty synthetically generated text files are used as input for the evaluation. These files are divided into three groups to reflect different entropy levels [7]:

- **Low-entropy files:** Composed of repeated character sequences to simulate compressible content.
- **Medium-entropy files:** Generated using a fixed vocabulary of common English words arranged randomly to mimic natural language text.
- **High-entropy files:** Contain uniformly random printable ASCII characters with minimal statistical redundancy.

Each file is stored in plain text format and ranges in size from 1 KB to 10 KB. This test set ensures a balanced evaluation across diverse data characteristics.

3.2 Evaluation Metrics

Three primary metrics are used to evaluate the performance of the compression tool:

- **Compression Ratio:** Defined as the ratio between the compressed and original file sizes, indicating space efficiency.
- **Correctness:** Determined by performing a byte-wise comparison between the decompressed output and the original file using the `diff` utility.

These metrics are also used to compare the performance of the implemented tool with `bzip2` and `xz` , which serve as representative benchmarks from established compression standards.

3.3 Experimental Procedure

For each of the thirty test files, the following steps are performed:

(1) The file is compressed using the implemented tool, as well as using `bzip2` and `xz` with default settings.
(2) The resulting compressed file sizes.
(3) The file is then decompressed using each respective tool
(4) The correctness of the implemented tool is verified by comparing the decompressed output against the original file.

All experiments are executed on the same machine under identical conditions. Intermediate files are removed after each iteration to avoid state leakage between runs. File sizes are recorded in bytes.

3.4 Result Interpretation

The results are analysed across multiple dimensions. Compression ratios are compared to determine the relative space efficiency achieved by each method. Special consideration is given to how the implemented tool performs relative to `bzip2` and `xz`, especially in the context of low- and medium-entropy data where the benefits of the BWT step are most pronounced.
Differences in compression effectiveness between entropy classes are also noted, providing insight into the types of input for which the custom tool is most suited. The absence of errors in decompression is taken as evidence of correctness and conformance with the requirements of a lossless compression pipeline.

4 RESULTS

This section provides an overview of the results obtain.

4.1 Compression Ratio

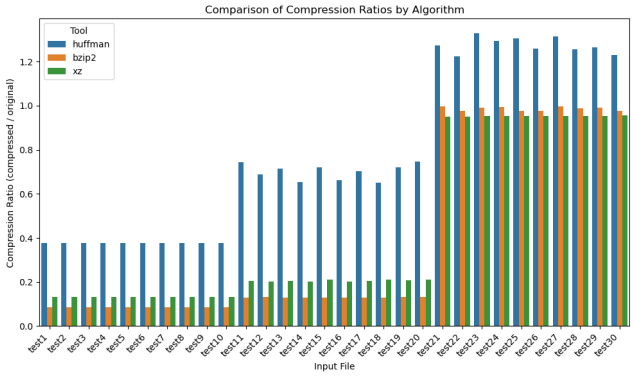

Figure 1: Figuring showing the compression ratio on various files for the different compression tools

The results in Figure 1 compare the compression ratios achieved by our custom Huffman-based tool (augmented with the Burrows–Wheeler Transform) against those of bzip2 and xz across a suite of 30 test files. For files 1–20, which likely contain structured or repetitive patterns and therefore exhibit lower entropy, the BWT-enhanced Huffman algorithm performs reasonably well, achieving compression ratios between approximately 0.36 and 0.75. This confirms that BWT successfully rearranges similar symbols to improve locality, enabling Huffman coding to exploit more favourable symbol distributions. However, even with BWT, the algorithm under performs relative to bzip2 and xz, both of which integrate additional transformations and modelling strategies such as Move-To-Front, Run-Length Encoding, or LZ77/LZMA-based compression. In files 21–30, which appear to have higher entropy (e.g., due to randomness, uniform symbol distribution, or prior compression), all three tools approach or exceed a compression ratio of 1.0. In this regime, Huffman coding is particularly disadvantaged: the BWT cannot introduce meaningful structure when none exists, and without further modelling layers, the compressor begins to expand the data. In contrast, bzip2 and xz maintain more efficient performance due to their layered compression strategies. These findings confirm that while BWT enhances basic Huffman coding, it remains sensitive to entropy levels and lacks the robustness of dictionary- or context-based methods under high-entropy conditions.

4.2 Correctness

The correctness was verified for all the test cases. There are no detected problems with the correctness of the implemented Burrow-Wheeler Huffman compression algorithm.

4.3 Statistical Ranking

Using a Friedman test it was determined that there is a statistically significant difference between the compression ratios of the three algorithms. Hence the ranking of the avg performance can be seen in the table below:

Table 1: Table showing average compression ratios and rankings

Tool	Avg Ratio	Score
BWT Huffman	0.783	3
bzip2	0.401	1
xz	0.431	2

Hence it can be seen that the algorithm presented in this paper performed the worst while the standard compression libraries performed very similarly although bzip2 did perform the best.

5 CONCLUSION

The development and evaluation of a BWT-enhanced Huffman compression tool demonstrated both the strengths and limitations of this classic approach. On structured, low-entropy data, the tool was able to achieve moderate compression ratios, confirming that the Burrows–Wheeler Transform successfully improves the input's compressibility by exposing redundancy. However, on higher-entropy data, the tool often failed to compress effectively and in some cases expanded the input, indicating a limitation in adaptability. Compared with bzip2 and xz, which employ more advanced modeling and transformation techniques, the implemented solution ranked significantly lower in compression efficiency. Statistical analysis using the Friedman test confirmed that these differences in performance are significant. Nevertheless, the implementation achieved perfect correctness across all test cases, and its modular structure makes it a useful reference or baseline for further development. Future improvements may include integration of additional modeling layers, such as Move-To-Front or LZ-based compression, to enhance performance on high-entropy data and close the gap with state-of-the-art compression tools.

REFERENCES

[1] Michael Burrows and David J. Wheeler. *A Block-sorting Lossless Data Compression Algorithm*. Technical Report 124. Palo Alto, CA: Digital Equipment Corporation, 1994.

[2] David A. Huffman. "A Method for the Construction of Minimum-Redundancy Codes". In: *Proceedings of the IRE* 40.9 (1952), pp. 1098–1101.

[3] Gonzalo Navarro and Veli Mäkinen. "Compressed Full-text Indexes". In: *ACM Computing Surveys* 39.1 (2007), Article 2. DOI: 10.1145/1216370.1216372.

[4] Igor Pavlov. *LZMA SDK (Software Development Kit)*. https://www.7-zip.org/sdk.html. Accessed: 2025-06-02. 2001.

[5] David Salomon. *Data Compression: The Complete Reference*. 3rd. Springer, 2004. ISBN: 978-0-387-40697-9.

[6] Julian Seward. "bzip2 and libbzip2: A Program and Library for Data Compression". In: *Free Software Foundation*. https://sourceware.org/bzip2/. 1998.

[7] Claude E. Shannon. "A Mathematical Theory of Communication". In: *Bell System Technical Journal* 27.3 (1948), pp. 379–423.