Bitlis Eren Üniversitesi Fen Bilimleri Dergisi
BİTLİS EREN UNIVERSITY JOURNAL OF SCIENCE
ISSN: 2147-3129/e-ISSN: 2147-3188
VOLUME: 12 NO: 3 PAGE: 783-796 YEAR: 2023
DOI:10.17798/bitlisfen.1301546

# Single and Binary Performance Comparison of Data Compression Algorithms for Text Files

Serkan KESKİN[1*], Onur SEVLİ[2], Ersan OKATAN[3]

[1] *Burdur Mehmet Akif Ersoy University, Institute of Science and Technology, Department of Computer Engineering, Burdur*
[2] *Burdur Mehmet Akif Ersoy University, Faculty of Engineering and Architecture, Department of Computer Engineering, Burdur*
[3] *Burdur Mehmet Akif Ersoy University, Gölhisar School of Applied Sciences, Department of Computer Technologies and Information Systems, Burdur*

*(ORCID: 0000-0001-9404-5039) (ORCID: 0000-0002-8933-8395) (ORCID: 0000-0001-6511-3450)*

**Keywords:** Text Compression, Data compression, Binary Compression, Deflate, BWT

**Abstract**
Data compression is a technique used to reduce the size of a file. To reduce the size of a file, unnecessary information is removed or parts that repeat the same information are stored once. Thus a lossless compression is achieved. The extracted file has all the features of the compressed original file and can be used in the same way. Data compression can be done using different techniques. Some of these techniques are Huffman coding, Lempel-Ziv-Welch coding and Burrows-Wheeler Transform. Techniques such as Huffman coding, Lempel-Ziv-Welch coding and Burrows-Wheeler Transform are some of them. Which technique to use depends on the type and size of the data to be compressed. Huffman, Lempel-Ziv-Welch, Burrows-Wheeler Transform and Deflate algorithms are the most widely used techniques for text compression. Each algorithm uses different approaches and can produce different results in terms of compression ratios and performance. In this study, different data compression techniques were measured on specific data sets by using them individually and in pairs on top of each other. The most successful result was obtained with the Deflate algorithm when used alone and the achieved compression ratio was 29.08. When considered in the form of stacked pairs, the compression ratio of the Burrows-Wheeler Transform and Deflate gave the best result as 57.36. In addition, when compression is performed in pairs, which algorithm is applied first and which algorithm is applied afterwards can make a significant difference in the compression ratio. In this study, the performance measurements obtained by applying the algorithms in different orders are compared and suggestions are presented to obtain optimum performance.

## 1. Introduction

The vast amount of data generated every moment is the basic building block of the digital world. Any information that can be measured and recorded can be referred to as data. This information can be in a wide variety of forms such as text, graphics, numbers, video, images and audio recordings. From personal files to the data centers of large companies, the amount of data collected and generated is enormous. Data can be obtained from a variety of sources [1]. For example, it can be entered as user input or collected automatically by software, sensors and devices. This collected data can be used for analysis and decision making. Information can often be stored on different platforms. Some are saved on the hard drive of computers, while others are stored on servers in cloud infrastructure [2].

* Corresponding author: serkankeskin@isparta.edu.tr
*Received: 24.05.2023, Accepted:04.09.2023*

In the 21st century, data is of great importance. Data is crucial for businesses, researchers, government agencies and even individuals.

However, storing and transmitting data comes with a number of disadvantages, such as increased volume and high costs [3]. As data capacities increase, storage and transmission costs also increase. This has led to the development of data compression techniques to reduce the data footprint. Data compression techniques have been developed to minimize storage space and reduce costs. It has also positively affected processing and analysis times. Data compression techniques encode information by compressing data into smaller formats, effectively reducing data size. The focus should be on using techniques to compress data in a performant way, thus making storage and transmission less costly [4]. This paves the way for faster and easily manageable transmission.

There are numerous approaches to data compression, determined by the degree of similarity between the compressed and original data, as well as the compression ratio. The history of compression techniques dates back to the advent of electronic digital computers. Early techniques were based on basic mathematical algorithms such as Huffman coding [5]. As technology advanced, lossless and lossy compression techniques emerged in the 1970s and 1980s. Later on, various techniques such as number-length coding, arithmetic coding and wavelet compression were developed. In addition to its effectiveness in storage and data transmission, compression has also become effective in data backup and data recovery. It has enabled the reduction of storage space for backups. Data compression is a widely used technique for archiving purposes. It has facilitated the storage of very large data for long periods of time while requiring minimal space.

### 1.1. Literature Review

There are different studies on data compression using many techniques to date. In a study conducted by Hasan in 2011, a compression study was carried out using Huffman and then Lempel-Ziv-Welch (LZW) techniques. A compression value of 3.25 was achieved on the data. When only one technique was applied, compression did not exceed 2.55 [3]. In another study conducted in the same year, the average compression ratio in multiple applications of the Huffman technique was 5.27 [6]. In the study conducted by Rahman and Hamada, the compression ratio of LZW technique was 1.28, Gzip technique was 1.5, LZMA technique was 1.32 and Brotli technique was 1.66. The original transform-based compression technique proposed in this study was found to be more successful than the other techniques with a compression ratio of 1.88 [7]. In another study conducted on texts based on LZW compression technique, it was concluded that the compression ratio remained at 1.33 [8]. In a study using Burrows-Wheeler Transform (BWT) and RLE techniques, the compression ratio remained at 2.48 [9]. In a study with Hybrid Sym6- Huffman coding, the compression ratio was 1.70 [10]. In a study comparing LZW, wavelet tree and compressed wavelet tree techniques, the compressed wavelet tree reduced a 200 KB file with a compression ratio of 4.65 [11]. In a study comparing the compression performance of Huffman and Unary coding on text files, Unary coding was found to be more successful. The compression ratio of the Unary code remained at 2.64 [12]. In a study by S. Kumar, a comparison was made between RLE (Run-length encoding) and ASCII encoding. The experimental study resulted in an average compression ratio of 2.53 [13]. A.Rahman compared Bzip2, Gzip, LZMA, Brotli and his proposed compression algorithms on 10 data sets. As a result of the comparison, Bzip2 algorithm was the most successful technique with a compression ratio of 2.91 [14]. P.Sarker, who performed compression with another text compression technique, achieved a compression ratio of 1.49 with his proposed technique [15]. S Haldar-Iversen performed binary compression with ASCII compression modulus+GZIP and obtained a compression ratio of 3.00 [16]. In a study on compression of dictionaries in different languages using the LZW algorithm, an average compression rate of 3.33 was obtained [1]. In a binary compression study with Chinese Remainder Theorem and Huffman algorithms, a success rate of 1.56 was achieved in license.txt text file [17]. In the study conducted by Ibrahim and Gbolagade, 4 different algorithms were used and the LZW algorithm with a compression ratio of 7.91 gave more successful results [18]. In the compression process performed with the help of a matrix table using the Huffman algorithm, a compression ratio of 2.94 was achieved in the

artificial text 1.txt text file [19]. In a study conducted by Rincy and Rajesh to examine the performance of the LZW algorithm with ASCII characters, a compression ratio of 4.23 was obtained [20]. In another study where the compression algorithm was constructed by utilizing long distance correlations between words, the compression ratio remained at 1.80 [21]. In this study, a new Karhunen-Loeve transform based algorithm for lossy image compression is developed, which presents a simple algorithm where images are only subsampled and KLT is applied. While most other image compression studies use hybrid methods, this study presents an approach based solely on KLT[22]. Ince et al. present the proposed LDR-DCT method as an alternative to the conventional DCT method when compression is unnecessary. It is also claimed that if the method is designed with quantisation tables, it can achieve the same JPEG image quality as the traditional DCT method and provide higher compression ratios [23]. In the study where data compression is performed by text clustering, the Compression Ratio Index (CRI), which can be calculated faster than internal methods such as Silhouette, Calinski-Harabasz and Davies-Bouldin indices, is developed. This study showed that SOI, an alternative clustering performance measure, gives consistent results with traditional internal and external methods [24].

## 1.2. Basic Principles of Data Compression

The process of reducing the footprint of electronic data is commonly known as data compression. Data compression is usually performed by two different techniques. These are redundancy removal and statistical coding techniques. These techniques help to optimize the storage of data on electronic devices. It is also possible to divide data compression into two parts: lossy and lossless.

### 1.2.1. Redundancy Removal

Redundancy removal is a valuable technique for removing repetitive or predictable data from datasets. In this technique, unnecessary spaces or characters are identified and removed, and is often used to compress text documents. This can minimize file size, save storage space and speed up data transfer [25]. There are three main methods for redundancy removal. The first is spatial redundancy, where similar or identical data is repeated within the same file. For example, the same pixels in a photo do not need to be repeated more than once. It is enough to save them once to reduce the file size. Redundancy can also be eliminated based on time. This relates to situations where the same or similar data is repeated at different points in time. For example, the same images appearing multiple times in a single video is redundant and requires more storage space. To avoid this, video sizes can be reduced by recording repetitive images once. The third and final redundancy removal technique is encoding redundancy. Encoding redundancy occurs when the same data uses more than one bit or symbol. By eliminating these redundancies, storage requirements can be reduced.

### 1.2.2. Statistical Coding

Statistical coding is based on the fact that some symbols or characters are more common than others in the dataset. There are two types of statistical coding. Entropy coding is a type of statistical coding that uses probability to assign variable length codes to symbols or characters. Lexical coding is a type of coding that replaces repeated words in a dataset with references to a dictionary or a table [5].

### 1.2.3. Lossless Data Compression

Lossless data compression is a technique used to compress data to take up less space. In this technique, no data is lost during the compression process. This means that the compressed data is exactly equal to the original data. Lossless compression algorithms usually compress by identifying duplicate parts of the data. These parts can significantly reduce the size of the data. For example, multiple repeated words or sentences in a text document can be recognized by lossless compression algorithms and stored in a smaller footprint. The most common use of lossless compression algorithms are file compression programs [19]. These programs save storage space by compressing particularly large files or multiple files together. Since compressed data has a smaller amount of data than the original data, there are fewer errors during data transmission. This form of compression is especially important in critical systems where there is no fault tolerance during data transmission. The most

common lossless compression algorithms are Huffman Coding, LZ77 and LZ78, BWT.

#### 1.2.4. Lossy Data Compression

When less memory or disk space is available to store information, lossy data compression is used to reduce the size of large data sets. However, this technique has a disadvantage. There is a possibility that some data may be lost during the compression process. Because of this potential loss, it is called lossy. This technique becomes more useful when working with multimedia files and other large data sets that require a significant amount of space to store and share. The size of files can be significantly reduced using lossy data compression techniques. Lossy compression is a technique that accelerates data compression by allowing efficient data storage and sharing. It can affect data quality while reducing storage requirements. Appropriate algorithm selection and proper configuration are required to achieve optimal results [26].

Different techniques are used in lossy data compression. Among the most important are volume-based, frequency-based and predictive techniques. Volume-based techniques evaluate the density and volume of data, such as how undetectable frequencies or low-density data can be bypassed by MP3 compression for audio files. Frequency-based techniques focus on frequency components using low frequency components to preserve essential information, such as JPEG compression that groups similar colors together and uses averages. Finally, predictive techniques identify recurring patterns in the data and efficiently reconstruct it using minimal information for patterns. For example, the GIF format can reduce the size of images by reusing similar colors in an image [18].
In short, lossless compression preserves all the original data, while lossy compression sacrifices some of the original data to achieve higher compression ratios. Lossless compression is typically used for text and data files, while lossy compression is typically used for image, audio and video files.

### 1.3. Areas Where Data Compression Is Used

#### 1.3.1. File Compression

The technique used to reduce file size is called file compression. Compression reduces the size of files on disk. Smaller files are easier to download, share and send. This allows users to save time, internet resources and storage space. File compression algorithms such as RAR, 7z, ZIP, GZIP are commonly used file compression applications.

#### 1.3.2. Video Compression

Video compression is the process of reducing the volume and flow rate of data and is used in direct relation to the term bandwidth. This compression technique works in the same way that a video camera captures each frame and converts it to JPEG format. If these frames are played back on a surveillance computer at 25 frames per second, you get moving video. This compression aims to provide high image quality. However, it also results in high bandwidth and storage overhead. Video compression algorithms are H.264, H.265, MPEG, HEVC, VP9, etc.

#### 1.3.3. Audio Compression

Audio compression is the process of fitting digitally recorded audio signals into a smaller volume with or without loss. FLAC, MP3, Ogg Vorbis, AAC are examples of popular audio compression algorithms.

#### 1.3.4. Image Compression

Image compression is a technique used to reduce the footprint of large image files. Generally, digital compression algorithms are used. These techniques are used to compress complex images such as photographs. JPEG, PNG, GIF are among the prominent ones of these techniques. In addition, high performance compression is performed with the discrete cosine transform (DCT) method. In order to minimise rounding errors and information loss, it is necessary to reduce the dynamic range of the DCT coefficients. In this way, a lower range of weights can be obtained according to frequency levels during DCT calculations [23].

## 2. Material and Method

### 2.1. Huffman Coding

Huffman coding is a commonly used technique in data compression. This technique reduces the data size by encoding frequently repeating symbols using fewer bits. This allows data to be

transmitted faster and uses less storage space. This algorithm first calculates the frequency of frequently repeated symbols and assigns shorter codes to these symbols. Rarely used symbols are assigned longer codes. In this way, the encoding of frequently used symbols uses fewer bits, while the encoding of infrequently used symbols uses more bits [27].

For Huffman coding, the symbols of the data are first identified and the frequencies of these symbols are calculated. These frequencies allow the symbols to be represented in a tree structure. Then, left branches in the tree structure are coded as 0 and right branches as 1. Since frequently used symbols will take shorter codes, the coding of these symbols will use fewer bits [15].

For example, if Huffman coding is done for the sentence "HELLO WORLD", the frequencies of the symbols of the text are first calculated. The letter "H" appears 1 time, "E" 1 time, "L" 3 times, "O" 2 times, "W" 1 time, R" 1 time and "D" 1 time. A Huffman tree is constructed according to the frequencies of these symbols [19]. First, the two lowest frequency symbols (here "E" and "R") are merged to form a node whose frequency is equal to the total frequency of the two symbols. This process continues according to the frequencies of the other symbols, and the tree structure is formed with the most frequently used symbol at the top. In this tree structure, a code is generated for each symbol. Frequently used symbols are assigned shorter codes, for example the symbol "L" is assigned a short code, while rarely used symbols are assigned longer codes. The Huffman coding for the sentence "HELLO WORLD" can be coded as shown in Table 1

**Table 1.** Huffman algorithm frequency and codes assigned to each symbol

| Symbol | Frequency | Code |
|---|---|---|
| E | 1 | 000 |
| H | 1 | 001 |
| D | 1 | 010 |
| R | 1 | 011 |
| W | 1 | 100 |
| O | 2 | 101 |
| L | 3 | 11 |

Table 1 shows the frequency of each symbol, the code assigned to the symbol and its path in the Huffman tree. For example, the code assigned to the symbol "L" is "11" and this symbol is located two nodes down the Huffman tree. Thanks to this coding technique, frequently used symbols in the text will receive shorter codes and the size of the text will be significantly reduced. For example, when Huffman encoding is used for the sentence "HELLO WORLD", the size of the text will decrease from 44 bits to 23 bits. This means that text can be transmitted and stored faster and using less storage space.

### 2.2. Lempel-Ziv-Welch

Among the data compression techniques currently in use, the LZW (Lempel-Ziv-Welch) algorithm is often preferred. It is a lossless technique. This means that no information is lost during the compression process. Basically, the algorithm identifies repeating patterns in the data and replaces them with shorter codes, resulting in compressed data [28]. Text files, graphics files and compressed data are typical applications of the LZW algorithm. Briefly summarizing the steps of the LZW algorithm: Initially, a dictionary is created by the algorithm consisting of codes for individual symbols such as "a", "b", "c", etc.

- During the compression process, individual data units are analyzed and the longest recurring pattern, also known as a word, is found. If the word is not found in the dictionary, it is given a code number and integrated into the dictionary.

- The word represented by the code number is added to the compressed data.
- Whenever a compressed data set contains a new term, the dictionary is immediately reviewed and a unique code number is assigned to the newly added word.
- This process continues in full until the piece of data is completely finished.
- The file format stores both the compressed data and a dictionary that allows the compressed data to be restored.

Using a window to identify patterns during compression, the LZW algorithm examines the data to detect repetition. The window size governs the pattern size for the search. For example, with a window size of 12 bits, the algorithm can identify up to 4096 unique words. The larger the window size, the longer patterns the algorithm can identify. However, this increases the complexity [20].
The LZW algorithm is particularly useful when dealing with data that contains repeating patterns, such as text files. In fact, it has proven effective in cases like "LLLLLLLLLLLLLLLLLLLLLLLLLLLLZ".
Instead of encoding "LLLLLLLLZ" every time it appears, the word is encoded only once and then represented by the corresponding code number each time it is repeated.

The LZW algorithm compresses the input data by replacing repeated patterns with shorter codes stored in a dictionary. The output of the algorithm consists of a set of indices corresponding to the codes in the dictionary. When the compressed data is decompressed, the dictionary is reconstructed using the same algorithm and the indices are replaced by the corresponding symbols in the dictionary [28].

As part of the compressed file, the LZW algorithm includes a table for code search. Overall, this table consists of 4,096 entries. The codes 0-255 in the table are assigned to represent individual bytes found in the input file. Before the initialization of the algorithm, only the first 256 entries of the table are filled, while the remaining entries are left blank. Basically, by default the first 256 codes are assigned to the standard character set. As the compression process evolves, the remaining codes are allocated to the sequences. During encoding initialization, the algorithm detects duplicate sequences in the data and adds them to the code table. It thus expands its content. In the context of file compression, codes between 256 and 4,095 are used to symbolize sequences of multiple bytes.

### 2.3. Burrows-Wheeler Transformation

The Burrows-Wheeler Transform (BWT) is an algorithm for text compression. This algorithm performs compression by identifying repetitive characters within a text. It is also based on the use of varying orderings of data based on their consecutive characters [9]. The stages of the BWT algorithm can be summarized as follows:

- Adding an EOF character at the end of the text: An EOF character is added at the end of the text.
- Creating all loops: All loops after the EOF character are created. Loops are created by shifting each character of the text to the right.
- Ordering of loops: All loops are sorted according to the lexicographic order of the characters in them.
- Creation of the BWT matrix: From each loop, the last character (except EOF) and all previous characters are copied into a matrix.
- BWT encoding: The characters of each column in the matrix are combined and used as the encoded representation of the compressed text. The key used during BWT encoding is the index of the last character in the original text. In this way, the compressed text can be reconstructed before it is encoded.

The BWT algorithm does not directly compress the data. Instead, it increases the compressibility of the data. The word "the" is most frequently used in the English text. Therefore, when the word "the" is encountered in the converted text, it is represented as "he". This feature has proven to be quite advantageous for various transformation algorithms, including Move-To-Front Transform [29]. When applying the BWT to an array, the resulting output of the Move Forward transform will consist mainly of smaller values that can be compressed efficiently using entropy encoding. We can rank the existing compression methods based on BWT in four different stages. The first stage includes the implementation of BWT, which serves as the core component of the compression algorithm. This operation increases the compressibility of the array. The next stage is known as global structure transformation (GST). At this stage, Burrows and Wheeler applied the Move Forward transformation as part of the list update algorithm [29]. The first version of the Burrows and Wheeler compression algorithm does not include a third step. However,

they introduced a concept that uses a code to symbolize the length of a string of zeros. In a later study, string length coding was applied as a tool to encode zero sequences and provided a commendable level of compression [30]. The final stage, the fourth stage, includes entropy encoding, which can be obtained through Huffman coding or arithmetic coding to compress the output of the previous stage.

The BWT algorithm performs compression by identifying repeating patterns. For example, in the text "bababababac", similar repeated characters could be "baba" and "c". The BWT algorithm creates blocks grouping the same characters and compresses by changing the order of the blocks. This can result in a significant reduction in data size.

The BWT algorithm is highly efficient for compressing data. However, in order to return the compressed data to its original format, the algorithm needs to know the index of the last character in the source text [31]. Furthermore, the compression efficiency of the algorithm may lag behind other existing algorithms in certain scenarios.

### 2.4. Deflate Coding

In mid-1990, Phil Katz developed a data compression format that preserves all original data, called lossless compression. This new algorithm is a combination of Huffman coding and LZ77 algorithms [24]. Deflate algorithm is a lossless compression algorithm used in compression programs such as gzip, PNG and WinZip. Data is compressed in consecutive blocks. Each block is compressed using Huffman coding and the LZ77 algorithm. The size of the compressible blocks varies and when the Huffman tree becomes too large for efficient coding, the Deflate algorithm terminates that block. It then starts a new block by creating a new Huffman tree. Each block consists of two parts. These parts are the compressed data and the Huffman code trees representing the data. In particular, the Huffman tree of each block is independent of the previous and the next block. The compressed data at the beginning of each block is preceded by Huffman trees compressed using Huffman coding. The LZ77 algorithm relies on a search buffer spanning 32,768 bytes and can refer to a string from the previous block as long as it stays within these limits. However, the length of the repeating character or forwarding buffer in this algorithm is limited to 258. The length of 256 different character numbers between 3 and 258 is represented as 1 byte. The 32,768-byte size search buffer is represented by 15 bits, while 1 bit is used for the flag representing the uncompressed data, so it is represented by 3 bytes.

### 2.5. Data Compression Performance

There are two parameters in data compression performance. These are data compression ratio and speed. Data compression ratio is expressed as the ratio of original data to compressed data. Take a 10 MB text or video file. Let the size of this file be 2 MB after compression. The compression ratio of this file is 5. An increase in the compression ratio means the direct success of the algorithm used. The other parameter, speed, refers to the compression time. As the speed increases, the time taken in the compression process will decrease. There is an inverse relationship between speed and compression ratio. As the speed increases, the compression ratio decreases. For this, it is important to achieve balance. The compression ratio is expressed by Equation (1).
Compression Ratio=original file/compressed file (1)

### 2.6. Data Set

In this study, 4 different data sets were used. The first data set is a text file named "pi.txt" consisting of the first one million digits of pi after the comma. This data set consists only of numbers and its size is 997 kilobytes. The second data set is a text file of the book "Alice's Adventures in Wonderland". It consists only of letters. The name of the data set is "alice.txt" and its size is 149 kilobytes. The third data set is a text file of firewall logs. This data set consists of 50% letters and 50% numbers. This data set is named "log.txt" and its size is 4.38 megabytes. The last data set is an Excel document. Like the log file, this document has a 50/50 ratio of letters and text. The size of the data set is 5.44 MB and is named "list.xls". "Alice.txt" was taken from the public domain [32]. The other datasets used were created by us.

## 3. Results and Discussion

Huffman coding, Deflate coding, LZW and BWT algorithms were used for data compression. With the algorithms, data compression operations were performed singly and sequentially in pairs. This data compression process was applied on 4 different data sets. Single compression rates and compression times are given in Table 2 and Table 3 respectively.

**Table 2.** Single data compression ratios

| Compression Ratios | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Data Set | Pi Dataset (996,19 kilobyte) | | Alice Dataset (148,52 kilobyte) | | Log Dataset (4283,64 kilobyte) | | List Dataset (5578.72 kilobyte) | |
| Algorithm | Compression The post File size | Compression Ratio | Compression The post File size | Compression Ratio | Compression The post File size | Compression Ratio | Compression The post File size | Compression Ratio |
| Huffman | 466,57 | 2,13 | 91,13 | 1,62 | 2732,83 | 1,57 | 5231.19 | 1.06 |
| LZW | 468,74 | 2,12 | 64,38 | 2,30 | 638,75 | 6,70 | 8042.12 | -1.69 |
| BWT | 444,73 | 2.24 | 48,18 | 3,08 | 577,45 | 7,41 | 1510.34 | 3.69 |
| Deflate | 486,06 | 2,03 | 53,66 | 2,76 | 147,3 | 29,08 | 1851.05 | 3.01 |

As seen in Table 2, the BWT algorithm is generally the most successful in single compression. This is due to the fact that it groups the same character blocks in the data sets and changes their order, which results in more successful results than other algorithms. The fact that the Deflate algorithm is more successful than the BWT algorithm on the log dataset can be explained by the fact that the dataset is more suitable for this algorithm. The negative result of the LZW algorithm on the list dataset is an indication that the algorithm cannot compress Excel files. It could not detect any similarity in the list dataset, thus increasing the character count instead of decreasing it.

**Table 3.** Single data compression times

| Compression Times (milliseconds) | | | | |
|---|---|---|---|---|
| Data set/ Algorithm | Pi Dataset | Alice Dataset | Log Dataset | List Dataset |
| Huffman | 313 | 187 | 1390 | 2492 |
| LZW | 157 | 78 | 267 | 802 |
| BWT | 4478 | 446 | 29540 | 40845 |
| Deflate | 120 | 86 | 98 | 198 |

Table 3 shows the compression times in milliseconds. The most successful algorithm in terms of compression time is the deflate algorithm. The reason why the deflate algorithm is more successful is that the compression is done by recording the initial position and length of the pattern. This data is written to the buffer as part of the compressed data. Thus, the algorithm compresses faster than other algorithms.

Although the BWT algorithm is more successful in single compression, it can be said that the deflate algorithm is more successful if we evaluate it together with the compression time. The compression time of the BWT algorithm is 206 times higher than the deflate algorithm for the list dataset, 301 times higher for the log dataset, 5 times higher for the alice dataset and 37 times higher for the pi dataset. The differences in compression ratios are not large. Based on the size of the datasets used, compression with BWT can be used for small datasets. However, for larger data sets, the BWT algorithm may take more time to compress. Considering the compression ratios and times together, it can be said that the deflate algorithm is more successful.

The binary compression results are detailed in Table 4 and Table 5.

**Table 4.** Compression ratios before and after Binary Compression

| Data Set | Pi Dataset | | Alice Dataset | | Log Dataset | | List Dataset | |
|---|---|---|---|---|---|---|---|---|
| Algorithm | 1.Compression ratio | 2.Post-compression ratio | 1.Compression ratio | 2.Post-compression ratio | 1.Compression ratio | 2.Post-compression ratio | 1.Compression ratio | 2.Post-compression ratio |
| Huffman+LZW | 2,135 | 1,634 | 1.629 | 1.390 | 1.567 | 2.410 | 1.066 | 0.767 |
| Huffman+BWT | 2,135 | 2.180 | 1.629 | 1.932 | 1.567 | 9.176 | 1.066 | 1.865 |
| Huffman+Deflate | 2,135 | 2.202 | 1.629 | 1.921 | 1.567 | 10.906 | 1.066 | 1.720 |
| LZW+Huffman | 2.125 | 2.100 | 2.306 | 2.274 | 6.706 | 6.664 | 0.693 | 0.770 |
| LZW+BWT | 2.125 | 2.123 | 2.306 | 2.310 | 6.706 | 18.704 | 0.693 | 2.427 |
| LZW+Deflate | 2.125 | 2.126 | 2.306 | 2.351 | 6.706 | 18.037 | 0.693 | 1.643 |
| BWT+Huffman | 2.239 | 2.243 | 3.082 | 3.159 | 7.418 | 29.499 | 3.693 | 4.375 |
| BWT+LZW | 2.239 | 1.631 | 3.082 | 2.395 | 7.418 | 44.950 | 3.693 | 3.501 |
| BWT+Deflate | 2.239 | 2.262 | 3.082 | 3.281 | 7.418 | 57.368 | 3.693 | 4.754 |
| Deflate+Huffman | 2.049 | 2.006 | 2.767 | 2.713 | 29.080 | 28.520 | 3.013 | 2.958 |
| Deflate+BWT | 2.049 | 2.048 | 2.767 | 2.753 | 29.080 | 29.214 | 3.013 | 3.013 |
| Deflate+LZW | 2.049 | 1.420 | 2.767 | 1.872 | 29.080 | 20.270 | 3.013 | 2.092 |

**Table 5.** File sizes in Kilobytes before and after binary compression

| Data Set | Pi Dataset (1020100 KiloByte) | | Alice Dataset (152089 KiloByte) | | Log Dataset (4386450 KiloByte) | | List Dataset (5712605 KiloByte) | |
|---|---|---|---|---|---|---|---|---|
| Algorithm | 1.File size after compression | 2.File size after compression | 1.File size after compression | 2.File size after compression | 1.File size after compression | 2.File size after compression | 1.File size after compression | 2.File size after compression |
| Huffman+LZW | 477767 | 624275 | 93321 | 109355 | 2798420 | 1819681 | 5356740 | 7444115 |
| Huffman+BWT | 477767 | 467909 | 93321 | 78696 | 2798420 | 478030 | 5356740 | 3061738 |
| Huffman+Deflate | 477767 | 463181 | 93321 | 79163 | 2798420 | 402183 | 5356740 | 3319809 |
| LZW+Huffman | 479988 | 485702 | 65931 | 66870 | 654076 | 658156 | 8235129 | 7417408 |
| LZW+BWT | 479988 | 480316 | 65931 | 65815 | 654076 | 234514 | 8235129 | 2353460 |
| LZW+Deflate | 479988 | 479776 | 65931 | 64668 | 654076 | 243180 | 8235129 | 3476828 |
| BWT+Huffman | 455408 | 454766 | 49345 | 48140 | 591313 | 148695 | 1546586 | 1305441 |
| BWT+LZW | 455408 | 625273 | 49345 | 63481 | 591313 | 97585 | 1546586 | 1631568 |
| BWT+Deflate | 455408 | 450954 | 49345 | 46350 | 591313 | 76461 | 1546586 | 1201640 |
| Deflate+Huffman | 497729 | 508283 | 54948 | 56059 | 150839 | 153801 | 1895478 | 1931159 |
| Deflate+BWT | 497729 | 498057 | 54948 | 55226 | 150839 | 150148 | 1895478 | 1895806 |
| Deflate+LZW | 497729 | 717965 | 54948 | 81240 | 150839 | 216396 | 1895478 | 2730094 |

In Table 4, the compression ratios of the first and second compression results of the experimental study with binary compression are given in three digits after the dot. Table 5 shows the size of the first and second compression files in kilobytes. In both tables, the most successful ones are bolded. In both tables, compression was performed using the two algorithms in succession. As a result of the compression process, the most successful result was obtained when using the BWT and Deflate algorithms respectively. The pi dataset, which is numeric data, exceeded the initial compression measurement by 0.97%. In the alice dataset, which consists only of letters, binary compression is better than the pi dataset. The Alice dataset shows an increase of 6.06% over the initial compression rate. For the List dataset, which is an Excel file, the situation is better. There is an increase of 22.3% over the initial compression ratio. In the Log dataset, which has the highest compression ratio, an increase of 87.06% over the initial compression ratio was realized. This is because the first compression increased the similarity and the second compression reduced more characters.

In the pi and alice datasets, the similarity was significantly reduced in the first compression. As a result, the second compression resulted in a low compression ratio due to low similarity. In the list dataset, the similarity was not reduced in the first compression and was compressed slightly more in the second compression.

The main issue here is the compression order. BWT-Deflate and Deflate-BWT do not have the same compression ratios. All compression algorithms ultimately achieve a certain amount of compression. However, when subjected to a second compression process, the two algorithms give different results. This is because the BWT algorithm sorts repeated characters as consecutive variables. This compresses to a certain extent. The Deflate algorithm compares the data in the buffer with the data in the window. If there is a pattern between the data in the window and the data in the buffer, it compresses that pattern. The data that was compressed in the first compression with BWT is made suitable for compression again with Deflate. This results in an increase in the compression ratio. However, when dual compression is performed as Deflate-BWT, the second compression cannot reduce the number of characters as a sequential variable, which is required by the BWT algorithm. Thus, the order of compression is important. This will also be valid for other algorithms. Table 6 shows the binary compression times.

**Table 6.** Binary compression times (milliseconds)

| Data Set | Pi Dataset | | Alice Dataset | | Log Dataset | | List Dataset | |
|---|---|---|---|---|---|---|---|---|
| Algorithm | 1.Compression time | 2.Compression time | 1.Compression time | 2.Compression time | 1.Compression time | 2.Compression time | 1.Compression time | 2.Compression time |
| Huffman+LZW | 313 | 49 | 187 | 14 | 1390 | 192 | 2492 | 514 |
| Huffman+BWT | 313 | 1696 | 187 | 318 | 1390 | 17617 | 2492 | 48597 |
| Huffman+Deflate | 313 | 37 | 187 | 12 | 1390 | 31 | 2492 | 132 |
| LZW+Huffman | 157 | 333 | 78 | 115 | 267 | 459 | 802 | 2994 |
| LZW+BWT | 157 | 1796 | 78 | 220 | 267 | 1897 | 802 | 67963 |
| LZW+Deflate | 157 | 22 | 78 | 11 | 267 | 32 | 802 | 211 |
| BWT+Huffman | 4478 | 230 | 446 | 126 | 29540 | 115 | 40845 | 541 |
| BWT+LZW | 4478 | 47 | 446 | 7 | 29540 | 16 | 40845 | 120 |
| BWT+Deflate | 4478 | 23 | 446 | 16 | 29540 | 12 | 40845 | 48 |
| Deflate+Huffman | 120 | 348 | 86 | 110 | 98 | 242 | 198 | 1108 |
| Deflate+BWT | 120 | 1789 | 86 | 172 | 98 | 496 | 198 | 13240 |
| Deflate+LZW | 120 | 59 | 86 | 6 | 98 | 17 | 198 | 297 |

Table 6 shows the binary compression times in milliseconds. In general, the Deflate-LZW binary compression algorithm performs the best in terms of binary compression times. If we look at the compression ratios, we can see that this has an inverse ratio. If we look at the BWT-Deflate binary compression algorithm, we can see that it has the highest compression time. From this we can conclude the following. The more compression, the more time it will take. Another aspect of compression time that we should not ignore is the type of data set. Text, numeric or text and numeric data also have different compression times.

As can be seen from Table 4 and Table 5, ranking in binary compression makes a big difference in both compression ratio and compression time. This is clearly seen in the experimental study. In general, the most successful result is obtained when BWT-Deflate algorithms are used consecutively.

In Table 7, the results of the study are tabulated in comparison with other studies in the literature.

**Table 7.** Similar Studies in the Literature

| Study Name | Algorithm Used | Compression Ratio |
|---|---|---|
| Hasan, 2011 [3] | Huffman+LZW | 3.25 |
| Hasan, 2011 [3] | LZH | 2.55 |
| Rahman and Hamada, 2020 [7] | Proposed Method | 1.88 |
| Barua et al, 2017 [8] | MLZW | 1.33 |
| Fruchtman et al, 2023 [9] | BWT+RLE | 2.48 |
| Amusa et al, 2022 [10] | Hybrid Sym6- Huffman coding | 1.70 |
| Gupta et al., 2022 [11] | Compressed wavelet tree | 4.65 |
| Wijaya et al., 2022 [12] | Unary Codes Algorithm | 2.64 |
| Kumar and Chatuverdi, 2021 [13] | RLE | 2.53 |
| Rahman and Hamada 2021 [14] | Bzip2 | 2.91 |
| Sarker and Rahman 2021 [15] | Proposed Method | 1.49 |
| Iversen, 2020 [16] | ASCII Compression Module+GZİP | 3.00 |
| Ignatoski at al., 2020 [1] | LZW | 3.33 |
| Ibrahim and Gbolagade, 2023 [17] | Huffman+CRT(Chinese Remainder Theorem) | 1.56 |
| Reza et al., 2019 [18] | Huffman | 2.30 |
| Bulut, 2016 [19] | Huffman | 2.94 |
| Rincy and Rajesh, 2019 [20] | LZW | 4.23 |
| Horspool and Cormack, 1992 [21] | UNIX Compress | 1.80 |
| **This study Pi Dataset** | **BWT+Deflate** | **2.26** |
| **This study Alice Dataset** | **BWT+Deflate** | **3.28** |
| **This study List Dataset** | **BWT+Deflate** | **4.75** |
| **This study Log Dataset** | **BWT+Deflate** | **57.36** |

Table 7 shows the success rates of similar studies in the literature and which algorithm is more successful. It is seen that the study is more successful than other studies.

## 4. Conclusion and Suggestions

In this experimental study, compression was performed on single and consecutive text files. In general, a certain amount of compression was achieved for all data sets and for all algorithms used. In the single compression process, the best results were obtained on the log data set. In this data set, it was observed that the Deflate algorithm achieved 96% data compression. One of the most successful results in terms of compression speed was achieved by compressing the log data set with the Deflate algorithm. The fact that the log data set is more successful than other data sets is due to the high similarity rate in the data set. For this reason, the Deflate algorithm was more successful than other algorithms in compressing log files.

Not all algorithms were successful in the dual, i.e. sequential compression process. Huffman-BWT, BWT-Huffman, Huffman-Deflate and BWT-Deflate algorithm pairs successfully compressed all data sets. The most successful result was obtained in BWT-Deflate dual compression. In the log data set, this success was achieved with a compression factor of 57.36. If we had performed the compression algorithm as Deflate-BWT in the log data set, this ratio would have been 29.21. The reason for achieving a higher compression ratio is that the blocks created by the BWT algorithm as a result of compression can be recompressed with the Deflate algorithm, which uses the tree structure. Therefore, it is very important which algorithm to use first in compression. Existing compression algorithms may have different sensitivity levels to different types of text data (e.g. news, articles, academic texts, poems, etc.). Thereby, we aim to perform sensitivity analyses to examine how compression performance varies according to the type of text. The development of compression algorithms optimized for specific types of texts is being considered.

### Conflict of Interest Statement

There is no conflict of interest between the authors.

### Statement of Research and Publication Ethics

The study is complied with research and publication ethics

## References


[1] M. Ignatoski, J. Lerga, L. Stanković, and M. Daković, ‘Comparison of entropy and dictionary based text compression in English, German, French, Italian, Czech, Hungarian, Finnish, and Croatian’, *Mathematics*, vol. 8, no. 7, p. 1059, Jul. 2020, doi: 10.3390/MATH8071059.

[2] I. B. Ginzburg, S. N. Padalko, and M. N. Terentiev, ‘Short Message Compression Scheme for Wireless Sensor Networks’, *Moscow Work. Electron. Netw. Technol. MWENT 2020 - Proc.*, Mar. 2020, doi: 10.1109/MWENT47943.2020.9067371.

[3] M. R. Hasan, ‘Data Compression using Huffman based LZW Encoding Technique’, *Int. J. Sci. Eng. Res.*, vol. Volume 2, no. 11, pp. 1–7, 2011, Accessed: Mar. 20, 2023. [Online]. Available: http://www.ijser.org

[4] V. Ratnam Anappindi, ‘Issue 8 www.jetir.org (ISSN-2349-5162)’, *JETIREZ06012 J. Emerg. Technol. Innov. Res.*, vol. 8, 2021, doi: 10.1109/EDSSC.2017.8126506.J.

[5] A. Habib, M. J. Islam, and M. S. Rahman, ‘A dictionary-based text compression technique using quaternary code’, *Iran J. Comput. Sci.*, vol. 3, no. 3, pp. 127–136, Sep. 2020, doi: 10.1007/s42044-019-00047-w.

[6] S. S and R. L, ‘Text Compression Algorithms - a Comparative Study’, *ICTACT J. Commun. Technol.*, vol. 02, no. 04, pp. 444–451, 2011, doi: 10.21917/ijct.2011.0062.

[7] M. A. Rahman and M. Hamada, ‘Burrows–wheeler transform based lossless text compression using keys and Huffman coding’, *Symmetry (Basel).*, vol. 12, no. 10, pp. 1–14, Oct. 2020, doi: 10.3390/sym12101654.

[8] L. Barua, P. K. Dhar, L. Alam, and I. Echizen, ‘Bangla text compression based on modified lempel-Ziv-welch algorithm’, *ECCE 2017 - Int. Conf. Electr. Comput. Commun. Eng.*, pp. 855–859, Apr. 2017, doi: 10.1109/ECACE.2017.7913022.

[9] A. Fruchtman, Y. Gross, S. T. Klein, and D. Shapira, ‘Weighted Burrows–Wheeler Compression’, *SN Comput. Sci.*, vol. 4, no. 3, pp. 1–12, Mar. 2023, doi: 10.1007/s42979-022-01629-5.

[10] K. Amusa, A. Adewusi, T. Erinosho, S. Salawu, and D. Odufejo, 'On the application of wavelet transform and Huffman algorithm to Yorùbá language syntax text files compression', *Serbian J. Electr. Eng.*, vol. 19, no. 3, pp. 351–368, 2022, doi: 10.2298/sjee2203351a.

[11] S. Gupta, A. K. Yadav, D. Yadav, and B. Shukla, 'A scalable approach for index compression using wavelet tree and LZW', *Int. J. Inf. Technol.*, vol. 14, no. 4, pp. 2191–2204, Jun. 2022, doi: 10.1007/s41870-022-00915-y.

[12] B. A. Wijaya, S. Siboro, M. Brutu, and Y. K. Lase, 'Application of Huffman Algorithm and Unary Codes for Text File Compression', *SinkrOn*, vol. 7, no. 3, pp. 1000–1007, Jul. 2022, doi: 10.33395/sinkron.v7i3.11567.

[13] S. Kumar and A. Kumar Chaturvedi, 'A Generalized Digital Database Text Compression Scheme Compared Wıth Ascii', *Int. J. Adv. Technol. Eng. Res.*, vol. 11, no. 2, p. 12, 2021, Accessed: Mar. 29, 2023. [Online]. Available: www.ijater.com

[14] M. A. Rahman and M. Hamada, 'Lossless text compression using GPT-2 language model and Huffman coding', *SHS Web Conf.*, vol. 102, p. 04013, 2021, doi: 10.1051/shsconf/202110204013.

[15] P. Sarker and M. L. Rahman, 'Introduction to Adjacent Distance Array with Huffman Principle: A New Encoding and Decoding Technique for Transliteration Based Bengali Text Compression', *Adv. Intell. Syst. Comput.*, vol. 1299 AISC, pp. 543–555, 2021, doi: 10.1007/978-981-33-4299-6_45.

[16] S. Haldar-Iversen, 'Improving the text compression ratio for ASCII text Using a combination of dictionary coding , ASCII compression , and Huffman coding', no. November, Nov. 2020, Accessed: Mar. 29, 2023. [Online]. Available: https://munin.uit.no/handle/10037/20517

[17] M. B. Ibrahim and K. A. Gbolagade, 'Performance Comparison of Huffman Coding and Lempel-Ziv-Welch Text Compression Algorithms With Chinese Remainder Theorem', *Univ. Pitesti Sci. Bull. Ser. Electron. Comput. Sci.*, vol. 19, no. 2, pp. 7–12, Dec. 2019, Accessed: Mar. 29, 2023. [Online]. Available: http://bulletin.feccupit.ro/archive/view/2019_2_2.html

[18] M. S. Reza, S. A. Riya, S. A. Alam, and M. A. A. Hossain, 'Study on Text Compression', Feb. 2019, Accessed: Mar. 29, 2023. [Online]. Available: http://dspace.uiu.ac.bd/handle/52243/822

[19] F. BULUT, 'Huffman Algoritmasıyla Kayıpsız Hızlı Metin Sıkıştırma', *El-Cezeri Fen ve Mühendislik Derg.*, vol. 3, no. 2, May 2016, doi: 10.31202/ecjse.264192.

[20] T. A. Rincy and R. Rajesh, 'Preprocessed text compression method for Malayalam text files', *Int. J. Recent Technol. Eng.*, vol. 8, no. 2, pp. 1011–1015, 2019, doi: 10.35940/ijrte.B1806.078219.

[21] R. N. Horspool and G. V. Cormack, 'Constructing word-based text compression algorithms', *Data Compression Conf. Proc.*, vol. 1992-March, pp. 62–71, 1992, doi: 10.1109/DCC.1992.227475.

[22] B. Eren, Ü. Fen, B. Dergisi, and S. Keser, 'An Image Compression Method Based on Subspace and Downsampling', *Bitlis Eren Üniversitesi Fen Bilim. Derg.*, vol. 12, no. 1, pp. 215–225, Mar. 2023, doi: 10.17798/BITLISFEN.1225312.

[23] I. F. Ince, F. Bulut, I. Kilic, M. E. Yildirim, and O. F. Ince, 'Low dynamic range discrete cosine transform (LDR-DCT) for high-performance JPEG image compression', *Vis. Comput.*, vol. 38, no. 5, pp. 1845–1870, May 2022, doi: 10.1007/S00371-022-02418-0/FIGURES/3.

[24] M. ASLANYÜREK and A. MESUT, 'Kümeleme Performansını Ölçmek için Yeni Bir Yöntem ve Metin Kümeleme için Değerlendirmesi', *Eur. J. Sci. Technol.*, no. 27, pp. 53–65, 2021, doi: 10.31590/ejosat.932938.

[25] R. Leelavathi and M. N. Giri Prasad, 'High-Capacity Reversible Data Hiding Using Lossless LZW Compression', *EAI/Springer Innov. Commun. Comput.*, pp. 517–528, 2022, doi: 10.1007/978-3-030-86165-0_44.

[26] J. R. Jayapandiyan, C. Kavitha, and K. Sakthivel, 'Optimal Secret Text Compression Technique for Steganographic Encoding by Dynamic Ranking Algorithm', *J. Phys. Conf. Ser.*, vol. 1427, no. 1, p. 012005, Jan. 2020, doi: 10.1088/1742-6596/1427/1/012005.

[27] M. M. Aşşık and M. Oral, 'Kanonik Huffman kod sözcükleri uzunluklarının evrim stratejileri algoritması ile belirlenmesi', *Gazi Üniversitesi Mühendislik-Mimarlık Fakültesi Derg.*, vol. 38, no. 2, pp. 771–780, 2022, doi: 10.17341/gazimmfd.882745.

[28] M. Varol Arısoy, 'LZW-CIE: a high-capacity linguistic steganography based on LZW char index encoding', *Neural Comput. Appl.*, vol. 34, no. 21, pp. 19117–19145, Nov. 2022, doi: 10.1007/s00521-022-07499-5.

[29] D. Zhang, Q. Liu, Y. Wu, Y. Li, and L. Xiao, 'Compression and indexing based on BWT: A

surveyZhang, D., Liu, Q., Wu, Y., Li, Y., & Xiao, L. (2013). Compression and indexing based on BWT: A survey. Proceedings - 2013 10th Web Information System and Application Conference, WISA 2013, 61–64. https://doi', *Proc. - 2013 10th Web Inf. Syst. Appl. Conf. WISA 2013*, pp. 61–64, 2013, doi: 10.1109/WISA.2013.20.

[30] P. M. Fenwick, 'The Burrows–Wheeler Transform for Block Sorting Text Compression: Principles and Improvements', *Comput. J.*, vol. 39, no. 9, pp. 731–740, Jan. 1996, doi: 10.1093/COMJNL/39.9.731.

[31] D. Kempa and T. Kociumaka, 'Resolution of the burrows-wheeler transform conjecture', *Commun. ACM*, vol. 65, no. 6, pp. 91–98, Jun. 2022, doi: 10.1145/3531445.

[32] 'Alice's Adventures in Wonderland dataset | Kaggle'. https://www.kaggle.com/datasets/roblexnana/alice-wonderland-dataset (accessed May 23, 2023).