Traditional Journal of Multidisciplinary Sciences (TJMS)

July- December 2023, Vol. 01, No. 02, [13 – 28]

Published: 10th August 2023

https://ojs.traditionaljournaloflaw.com/index.php/TJMS

RESEARCH PAPER

Implementation and Analysis of Lossless Data Compression

Khaqan Arif[1] Syed Abdur Raheem Ali shah[2] Amad Ur Rehman[3] Alamzeb[4]

[1-2] Department of Electrical Engineering Hamdard University Islamabad, Pakistan.
[3] Department of information and network security, Beihang University of Aeronautics and Astronautics, China.
[4] Faculty of Engineering Science and Technology, Hamdard University, Islamabad, Pakistan.

**Corresponding Author* khaqanarif0@gmail.com

ABSTRACT

The paper focuses on the invention and examination of three lossless data compression techniques: Shannon-Fano, Huffman, and the Burrows-Wheeler Transform (BWT). The results of each approach applied to a group of files are compared and evaluated. The Shannon-Fano approach has varied compression and decompression timings across files, with compression timings ranging from 30 to 260 milliseconds and decompression timings ranging from 120 to 2000 milliseconds. The approach achieves varying levels of compression, resulting in less storage space and computing time among datasets. The Huffman technique, on the other hand, yields compression ratios that vary between 0.19 to 0.9, with associated saving percentages ranging from 20.0% to 72.5%. The BWT algorithm showcases compression ratios ranging from 0.2 to 0.7 and saving percentages from 25.0% to 90.0%. By analyzing the results, the study provides valuable insights into the performance of these compression techniques and their effectiveness in handling diverse datasets.

Keywords: Compression Techniques, Huffman Coding, Shannon-Fano Algorithm, Burrows-Wheeler Transform (BWT).

INTRODUCTION

The development and generation of digital information have reached unprecedented levels in today's data-driven society, embracing multiple fields like communications, storage, and applications for multimedia (Fitzgerald, 2020). With the growing development of data, the necessity for effective data storage and transfer technologies is becoming increasingly important (Ahmad et al.,2022). Data compression, an important method in computer science and signal processing (Russell & Wang, 2022), tackles this difficulty by shrinking data without surrendering any information, allowing for more efficient and possible resource utilization and improving overall system's efficiency (Wright & Ma, 2022). A compressed photo requires fewer bits than its uncompressed counterpart, so it is transmitted faster, and your hardware can process it more quickly; ultimately, the photo loads faster in your browser (Umbaugh, 2022). An audio and video file can be compressed by up to 90%, so you can stream it all over the world within seconds (Seeliger et al., 2022). Compressed images, videos, and audio files on mobile devices are transferred to cloud servers faster, which saves you time when you back up your devices (Anaspure, 2022). Lossless data compression algorithms have emerged as a viable solution, with the ability to properly rebuild the original data without information loss (Nassra & Capella, 2023). Lossless compression methods, as opposed to their lossy equivalents, are especially important in situations when data accuracy is critical, including scientific data, healthcare records, monetary transactions, and preservation purposes (Gudodagi et al.,2023).

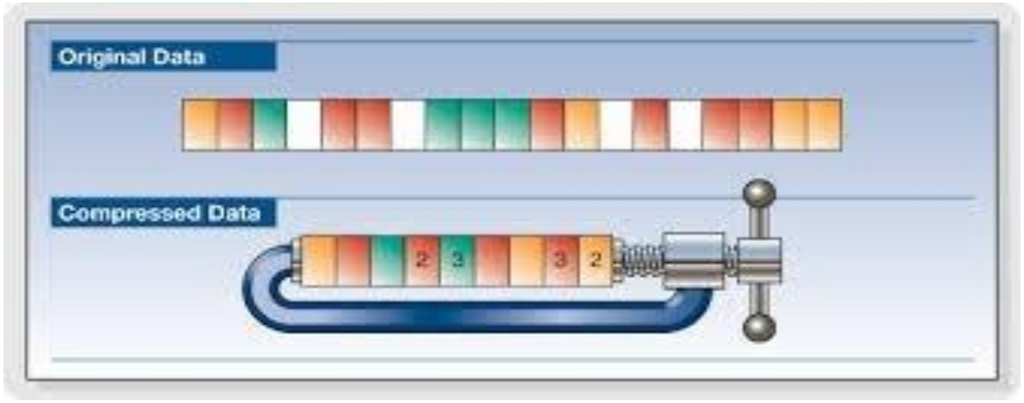


Fig 1.1: Data compression

This research article focuses on the implementation and study of lossless compression of data techniques. As the demand for fast data handling grows, it is critical to investigate and assess the performance of different compression algorithms in order to determine their usefulness across diverse types of applications and data. This paper intends to contribute to a better knowledge of lossless compression of data by shining light on its advantages, limitations, and optimisation possibilities. The results of this research paper are mainly intended to make a substantial contribution to the study of lossless compression of data by directing the selection and execution of appropriate compression techniques for a variety of real-world applications. We hope to enable technologies and industries that depend on massive amounts of data by improving compression performance and reducing data redundancy, bringing us closer to a more interconnected and efficient digital society. In our paper, we study different methods of lossless text data compression algorithms and calculating the compression size, compression ratio, processing time or speed using Shannon- Fano, Huffman and Burrows Wheeler Transform Coding.

Finally, the outcomes of this paper will help to provide a complete and specific comprehension in lossless data compression techniques, their practical practicality, and their possible uses for optimizing the transmission and storage of data in a variety of real-world circumstances. The findings of this research will also help data practitioners, programmers, and academics choose the best compression algorithms based on the type of their data as well as the performance needs of their systems.

LITERATURE REVIEW

The following review of the literature provides a review of the major research papers and advances in the execution and evaluation of lossless compression of data techniques.

Several studies (Biagetti et al.,2021) (Otair et al.,2022) (Hidayat et al.,2020) have been conducted to compare the performance of various lossless compression techniques. Mohammadi et al. (2022) compared Huffman coding, LZW, or the BWT algorithms on diverse data formats, highlighting their advantages and disadvantages. Al-Qurabat and Kadhum (2021) compared lossless compression algorithms and analysed their usefulness for various application scenarios. Researchers also investigated the adaptation of lossless compression techniques to various data formats (Zhang et al.,2023). Long et al. (2021), for example, studied the performance of several important methods on genomic data, whereas Mallik and Zhao (2020) investigated their usefulness in compressing images from medical imaging. Arithmetic coding is a statistical encoding technique proposed in 1987 by researchers Ian H. Witten & Radford M. Neal that specifically provides fractional bit illustrations for symbols. (Mentzer et al.,2020) It allocates each symbol a unique interval based on its likelihood of occurrence, leading to concise representation for commonly seen symbols. Townsend (2021) investigated theoretical elements of arithmetic coding, whereas later research concentrated on hardware solutions for quicker decoding and encoding applications. (Ma et al., 2019). The Burrows-Wheeler Transform, invented by Michael Burrows & David J. Wheeler in 1994, is a data transformation tool (Bello, 2020). When paired with a move-to-front and run-length encoding, it organizes the input data to produce runs of identical characters, resulting in greater compression ratios (Kumar et al., 2019). Rahman & Hamada (2020) recently investigated the application of simultaneous processing and acceleration in hardware to optimise the BWT compression technique. The LZW algorithm, developed by Abraham Lempel, Jacob Ziv, and Terry Welch in 1984, is a dictionary-based compression method. It builds a dictionary of variable-length codes from the input data and effectively represents repetitive patterns with shorter codes. The LZW algorithm has been extensively studied and widely implemented in various compression utilities, with research focusing on adaptive variations and dictionary management techniques (Rahman & Hamada, 2019; Shah & Banday, 2020). David A. Huffman created Huffman coding in 1952, and it is one of the first and most extensively used lossless compression algorithms (Karim et al., 2021) (Erdal & Ergüzen,2019). By considering the frequency of the presence of symbols within the input data, the algorithm generates a variable-length prefix code. Sarangi (2022), for example, investigated several tweaks and optimizations to increase the compression effectiveness of Huffman coding. SANDHU (2021) also presented a distributed implementation of the Huffman method that takes advantage of multi-core CPUs to achieve higher compression speeds.

RESEARCH METHODOLOGY

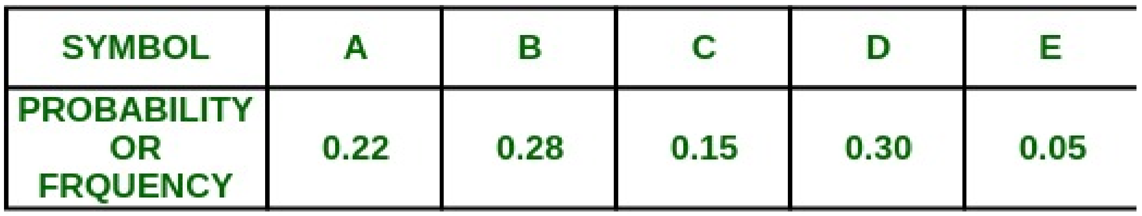
Algorithms Applied

Shannon-Fano: Around 1960, Claude E. Shannon from MIT and Robert M. Fano from Bell Laboratories jointly developed a coding procedure that led to the creation of a binary code tree

(Williams, 2022). This innovative procedure involved evaluating the probabilities of symbols and assigning code words based on their respective code lengths. Although the Shannon-Fano coding technique is very simple and easy to develop in comparison to other ways, but it also lacks practical significance due to lesser code effectiveness when compared with Huffman coding, as demonstrated in future demonstrations. (Lawal et al., 2021). The procedure of building a Shannon-Fano tree adheres to a certain specification to establish a successful code table, and the technique itself is straightforward. To build the code table, a listing of symbols and their accompanying percentages or frequency counts are created and implemented, allowing the relative frequency of occurrence of each symbol to be determined accurately. Despite its historical relevance, the Shannon-Fano coding methodology has seen limited implementation in practise, with more effective approaches such as Huffman coding dominating in a variety of data compression applications. (Oswald & Sivaselvan, 2023; Grewal, 2021).

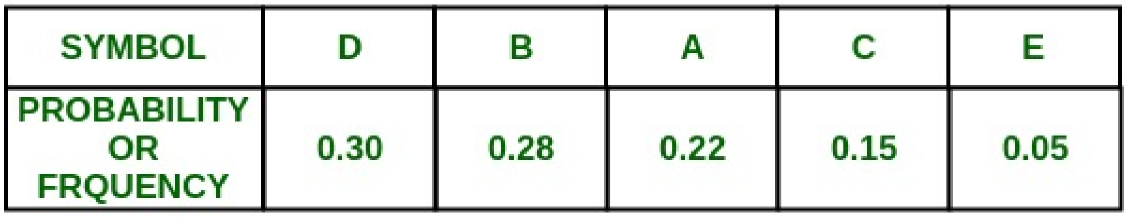
Shannon-Fano Algorithm:

1. Sort the symbol lists by the level of frequency, with the ones that appear most frequently appearing symbols on the left side and the least frequently occurring symbols on the right side.
2. Split the list into two main sections, with the overall frequency numbers for the left part or side closest to the overall frequency values for the right side as appropriate.
3. The binary digit 0 is then assigned to the left portion of the list, and the binary digit 1 is assigned to the right part of the list.
4. Apply and repeat steps 3 and 4 recursively to each of the two portions, subdividing groups and inserting bits to the code until and unless every character has become a code leaf on the tree.



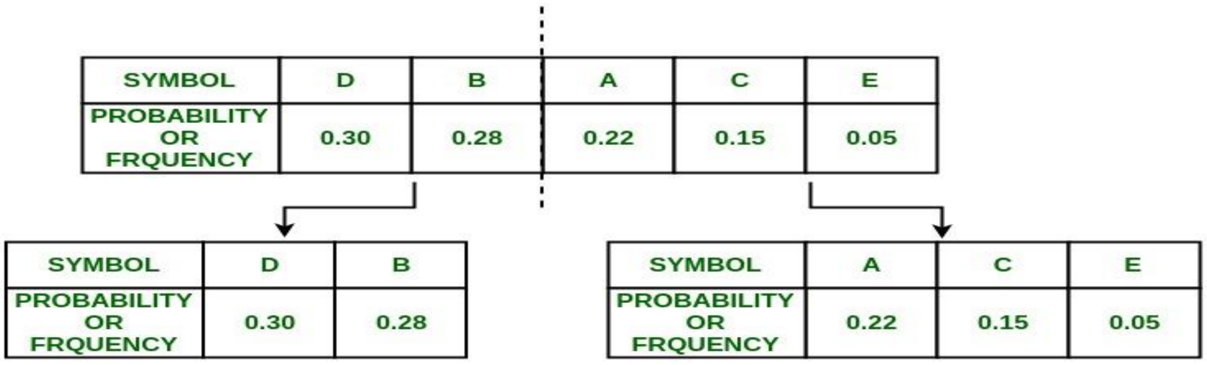
SYMBOL	A	B	C	D	E
PROBABILITY OR FRQUENCY	0.22	0.28	0.15	0.30	0.05

Step 1

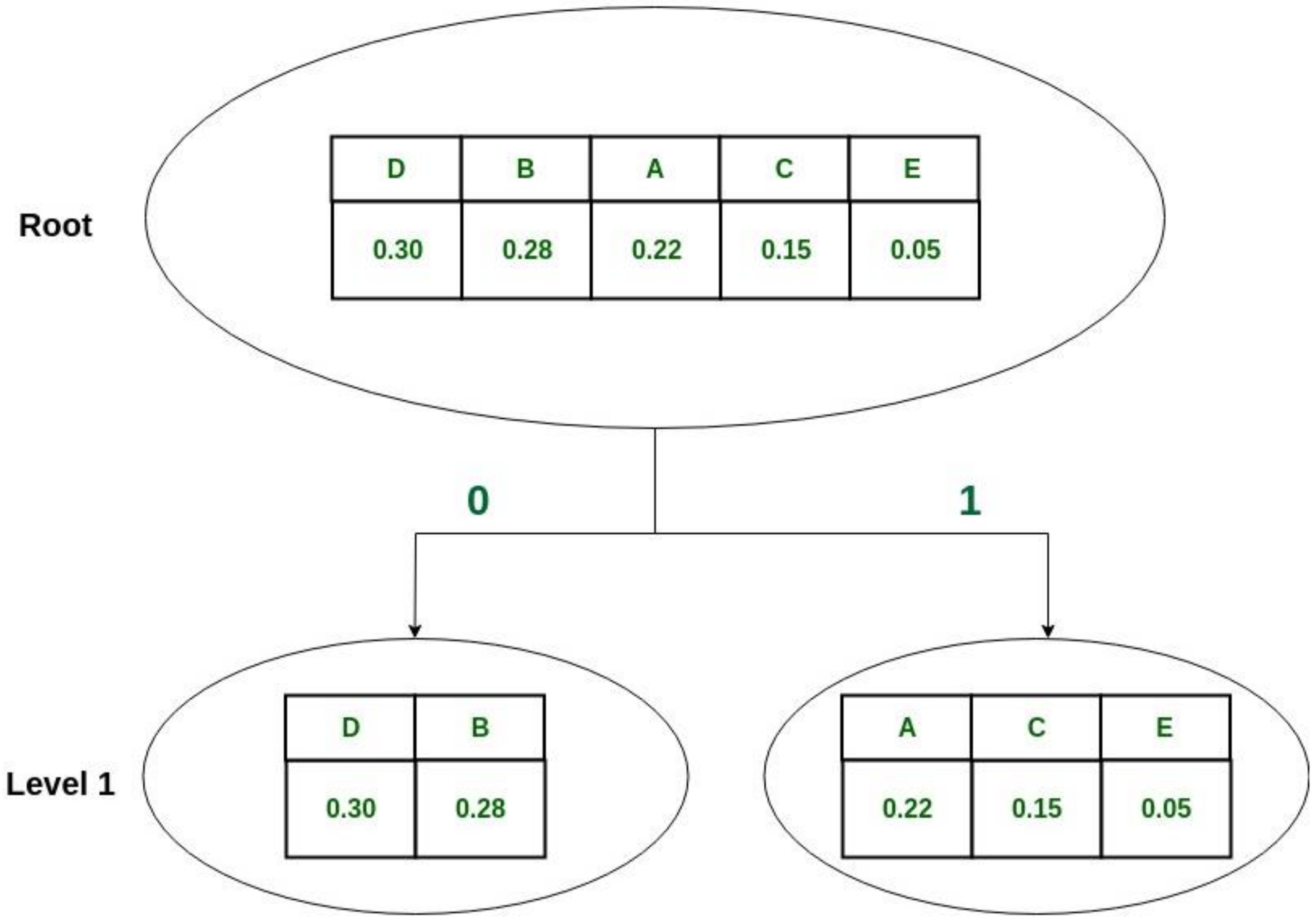


SYMBOL	D	B	A	C	E
PROBABILITY OR FRQUENCY	0.30	0.28	0.22	0.15	0.05

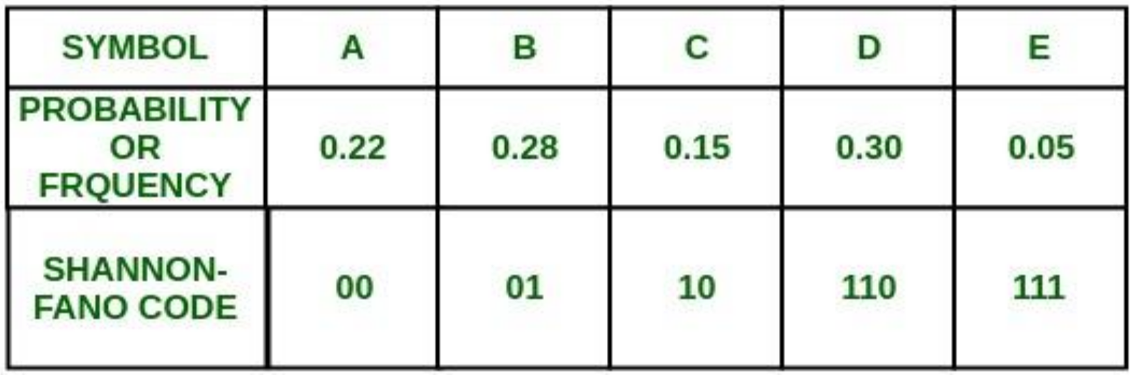
Inputs are sorted according to their Frequency.



Step 2: Symbols are divided into two such that sum of the probability on the left side is almost equal to the probability on the right side (Repeat for all the symbols).



Step 3 (It will repeat for all symbols)



SYMBOL	A	B	C	D	E
PROBABILITY OR FRQUENCY	0.22	0.28	0.15	0.30	0.05
SHANNON-FANO CODE	00	01	10	110	111

Final Step

Huffman Encoding

Huffman coding is an entropy encoding algorithm for lossless statistics compression (Liu et al.,2022). In this set of rules, many fixed length codes are being replaced with the aid of variable-length codes. While using variable-length code phrases, it is suited to create an expected prefix code, avoiding the need for a separator to determine codeword barriers or chellenges. Huffman Coding mostly uses such a prefix code. A specific probability distribution is used to build a code tree using Huffman coding. (Nosratian et al.,2021). It varies in three types (Ashila et al., 2019).

1. Static Probability distribution
2. Dynamic Probability distribution
3. Adaptive Probability distribution

Huffman Algorithm: The input is an array of unique characters accompanied by their frequency of occurrences, and the output is a Huffman Tree based on the given input. (Moffat, 2019) (Kumari & Saini, 2022)

1. Make a (leaf) node for every particular symbol and generate a min-heap by the leaf nodes
2. Find nodes which have the nominal rates from the min heap.
3. Construct a new innermost node with the same number of occurrences as the sum of the frequencies of the other two nodes. Create the initial extracted node the left node & the second the right node. Add the following node to the bottom of the heap.
4. Steps 2 and 3 should be repeated until the heap holds only one node. The final node is a foundation node, and the structure of the tree is complete.

Flow of Huffman Coding:

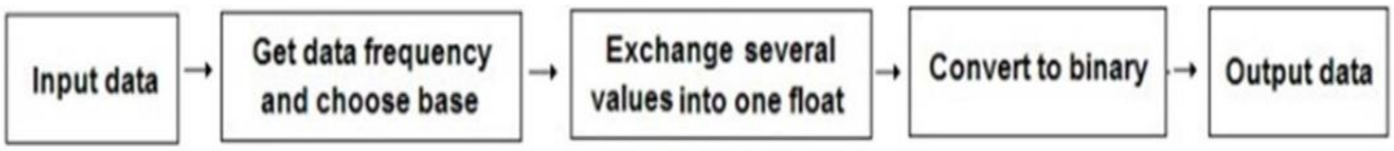

Input data
Get data frequency and choose base
Exchange several values into one float
Convert to binary
Output data

Fig.2.2 Data compression Flow Graph

I. Symbols that appear frequently use lower encoding than symbols that appear less frequently.

II. Both symbols that appear the fewest times will be the same length. The Huffman algorithm employs the greedy method, in which the collection of rules selects the most pleasurable choice at each step. A binary structure is built from the ground up. Let us look at an example of how Huffman Coding works. Expect the following frequencies for each character in a compressed report:

A: 25 B: 10 C: 99 D: 87 E: 9 F: 66

The method of making this tree is:

1. Create an extensive list of leaf nodes for each image and arrange the nodes according to the order of descent.

   C: 99 D: 87 F: 66 A: 25 B: 10 E: 9

2. Choose the leaf nodes that have the lowest frequency. Construct a parent node containing both of these nodes and apply the same frequency to the total of the two infant node frequencies.

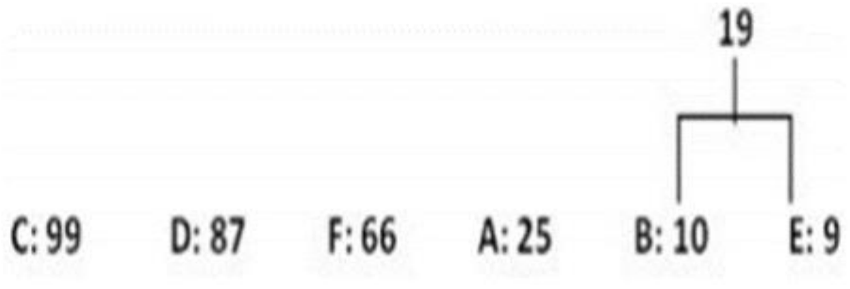


Fig.2.3 Lowest Parent Node Creation

3. Now, inside the list, add the determined node and remove the two baby nodes. Repeat this procedure until you only have one effective node remaining.

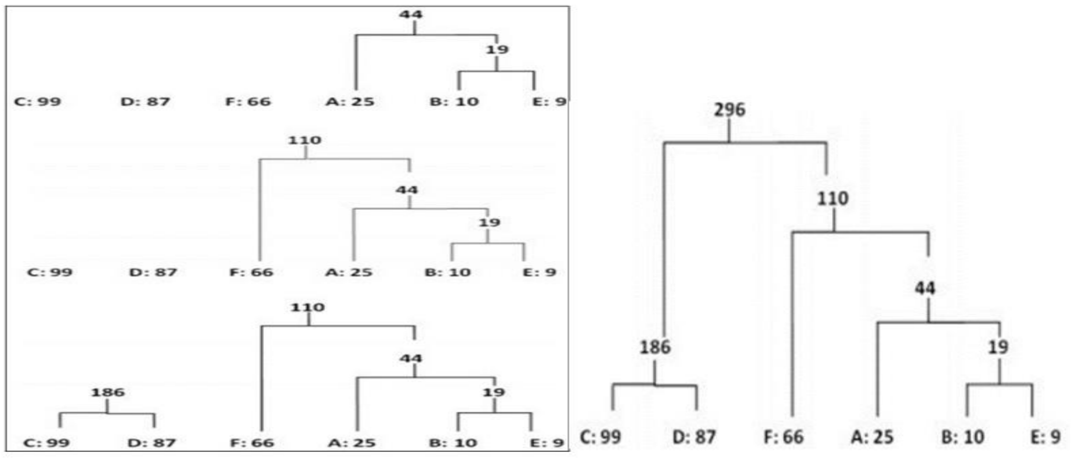
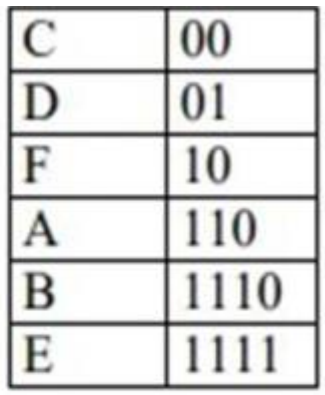


Fig.2.4 Huffman Code Tree

4. Now label every aspect. The left child of every parent is categorized with the digit 0 and right toddler with 1. The code phrase for each source letter is the order of labels alongside the direction from root to the leaf node in place of the letter.

Huffman Codes are proven beneath within the table:



C	00
D	01
F	10
A	110
B	1110
E	1111

Burrows Wheel Transform Coding

In 1994, M. Burrows and D. Wheeler presented a new data compression technique based on a pre-processing on the input string. Such a Pre-processing, called after them the Burrows Wheeler Transform (BWT), produces a permutation of the letters in the input string such that (Giuliani et al.,2021) (Begum et al.,2023)

a. the transformed string is easier to compress than the original one.
b. the original string can be recovered.

It is a lexicographically reversible permutation of a string's characters. It is the first of three steps to be taken in order when developing the Burrows-Wheeler Data Compressed algorithm, which is the foundation of the Unix compression programme bzip2.The Burrows-Wheeler Transform offers two decoding algorithms.

1. The first relies on adding and sorting the output text to make a number matrix M of N x N dimensions in order to revert the coding process. This approach is simple and straightforward, but it requires more time for the computer to complete, and the original text S is not recovered directly.
2. The second approach is more complex than the first depending on the permutations, but it requires less commuting time as well as storage and will be described. Unlike the first technique, the result of this method corresponds to the original string S; at the start, we have the converted string L as well as the index I.

To encode using MTF, a dictionary which is ordered according to the symbols being entered must be constructed. This is accomplished by shifting the element in the list of symbols that corresponds to the content of the symbol to the top of the dictionary. Although the preceding method is used for encoding, it is also used for decoding.

Burrows Wheeler Transform Algorithm

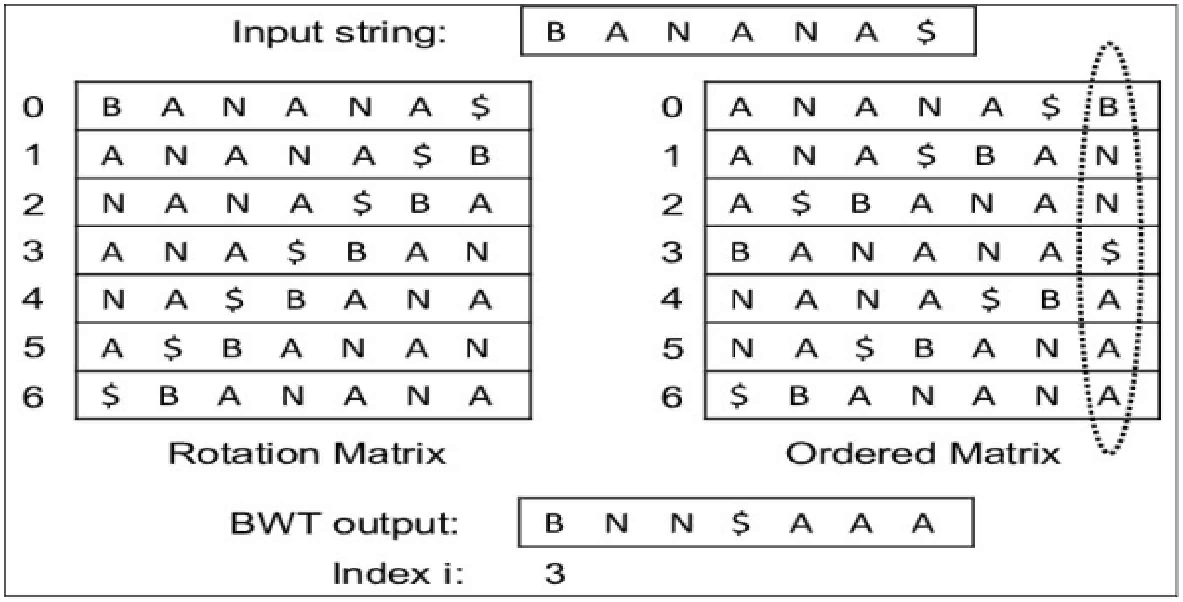


Fig.2.5 BWT Coding

Data Collection

Text data with the extension txt was utilised as an example in evaluating this compression strategy. Three text examples files were used in this investigation. Each algorithm has its own:

NO	**Compression Technique**	**.txt Size in Bytes**
1.	Shannon-Fano Algorithm	13500
2.	Huffman Coding	15600
3.	Burrows Wheel Transform Coding	10600

Table 1 Algorithm's Data Collection

Analysis Technique

In this comparison, several types of analysis were used: (Sharma & Batra, 2020).

1. The first step is to analyse how it operates in addition to the algorithms.
2. The first step is to do an analysis of how it's done as well as a study of algorithms.
3. Compression time = Starting time - Finishing time

Flow of Data Collection and Analysis

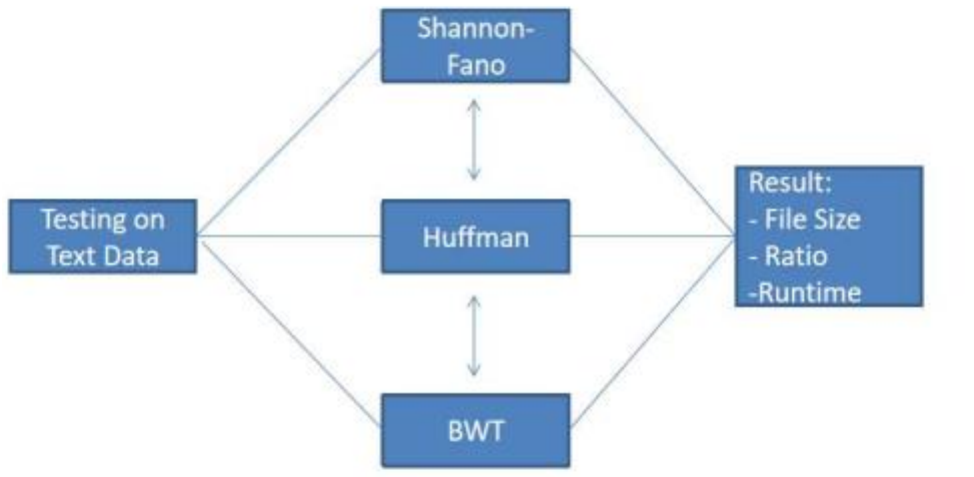


Fig 2.6 Data Flow

RESULTS AND FINDINGS

Shannon-Fano Algorithm

The results reveal that compression and decompression durations vary between files, from 30 to 260 milliseconds for compression and 120 to 2000 milliseconds for decompression. Overall, the result shows the Shannon-Fano algorithm's success in achieving various levels of compression, reducing storage space, and the related choices with computational time required across various datasets.

S. No	File Size (Bytes)	Comp File Size (Bytes)	Compression Ratio	Compression Factor	Saving Percentage	Comp Time (mS)	Decomp Time (mS)
1	204800	102400	0.5	2.0	50.0%	120	700
2	128000	45056	0.352	2.84	64.8%	90	450
3	65536	40960	0.625	1.6	37.5%	50	280
4	409600	122880	0.3	3.33	70.0%	200	2000
5	256000	51200	0.2	5.0	80.0%	140	900
6	10240	7680	0.75	1.33	25.0%	30	120
7	81920	53248	0.65	1.54	35.0%	180	800
8	86016	33792	0.39	2.96	61.0%	80	400
9	163840	94208	0.575	1.74	42.5%	260	1500
10	28672	15360	0.535	1.87	46.5%	100	550

A compression ratio of 1.0 indicates that no compression was performed, but lower numbers suggest more efficient compression (Gui et al.,2019) This information is supplemented by the "Compression Factor" column, which basically represents the inverse of the compression ratio. A compression factor of 2.0, means that the compressed file is double as small as the original, but a value of 1.33 means that the compressed file is about 1.33 times smaller. Overall "Saving Percentage" column shows the percentage of file size reduction accomplished through compression, with values ranging from 25.0% to 80.0%. Higher reduction in space and more efficient compression are indicated by higher saving percentages.

Huffman Algorithm

S. No	**File Size (Bytes)**	**Comp File Size (Bytes)**	**Compression Ratio**	**Compression Factor**	**Saving Percentage**	**Comp Time (mS)**	**Decomp Time (mS)**
1	204800	102400	0.9	2.3	20.0%	80	300
2	128000	45056	0.215	2.14	54.8%	100	250
3	65536	40960	0.545	1.0	32.5%	210	270
4	409600	122880	0.2	2.33	60.0%	300	1000
5	256000	51200	0.3	6.0	38.0%	150	800
6	10240	7680	0.55	1.23	55.0%	60	110
7	81920	53248	0.76	1.24	35.0%	130	300
8	86016	33792	0.19	1.83	41.0%	70	500
9	163840	94208	0.525	1.72	72.5%	250	1600
10	28672	15360	0.545	1.97	56.5%	110	150

The table shows the results of applying the Shannon-Fano algorithm to a set of files for lossless data compression. The "Compression Ratio" column displays the original file size to compressed file size ratio, which ranges from 0.19 to 0.9. Lower compression ratios indicate that compression is more efficient. The "Compression Factor" column displays the reciprocal of the compression ratio, with values ranging from 1.0 to 6.0, with larger compression factors indicating better compression efficiency. The "Saving Percentage" column displays the proportion of file size reduction accomplished using compression, which ranges from 20.0% to 72.5%. Higher saving percentages indicate greater space savings.

Burrows Wheel Transform Algorithm

S. No	**File Size (Bytes)**	**Comp File Size**	**Compression Ratio**	**Compression Factor**	**Saving Percentage**	**Comp Time (mS)**	**Decomp Time (mS)**

		(Bytes)					
1	204800	102400	0.6	2.0	25.0%	120	400
2	128000	45056	0.3	2.74	50.8%	80	450
3	65536	40960	0.4	2.6	70.5%	50	180
4	409600	122880	0.3	2.33	40.0%	100	3000
5	256000	51200	0.2	4.0	90.0%	140	400
6	10240	7680	0.5	1.53	50.0%	40	110
7	81920	53248	0.6	2.14	35.0%	150	600
8	86016	33792	0.3	2.76	31.0%	40	500
9	163840	94208	0.7	1.25	52.5%	230	1200
10	28672	15360	0.5	1.27	66.5%	300	120

The table clearly shows the results of applying the Shannon-Fano algorithm to a set of files for lossless data compression. Each row corresponds to a specific file, and the table gives many critical metrics to evaluate the compression process's effectiveness. The "Compression Ratio" column displays the original file size to compressed file size ratio, which basically ranges from 0.2 to 0.7. Lower compression ratios indicate that compression is more efficient. The column "Compression Factor" reflects the reciprocal of the compression ratio, which ranges from 1.25 to 4.0. However, higher compression factors indicate improved compression efficiency. The "Saving Percentage" column shows the percentage of file size reduction accomplished using compression, which ranges from 25.0% to 90.0%. Higher saving percentages indicate greater space savings.

During our research on text compression algorithms, we conducted implementations of various methods and drew important conclusions from our experiments. Specifically, we compared the performance of the Shannon-Fano algorithm with the Huffman compression technique. Despite both algorithms employing similar compression processes, we found that the Shannon-Fano compression exhibited less efficiency when compressing text files compared to Huffman compression. This was evident from the compressed file size, where Shannon-Fano yielded larger files than Huffman, consequently resulting in a lower compression ratio for Shannon-Fano. Moreover, the compression time for Shannon-Fano was significantly higher, taking approximately 67.077 seconds, while Huffman only took 1.313 seconds. Comparatively, the BWT (Burrows-Wheeler Transform) algorithm proved to be the fastest, completing compression in 0.1522 seconds. These findings highlight the trade-offs between the different compression algorithms, with Huffman emerging as the more effective and efficient option for text compression in our experiments. Furthermore, the computational challenge of Huffman coding remained low, allowing for fast encoding and decoding procedures. This efficiency is especially beneficial in circumstances requiring real-time or quick data compression and decompression, such as communication networks or high-throughput data transmission systems. It should be emphasized, however, that Huffman coding could not always be the best option for all sorts of data. When working with data that lacks clear patterns or when confronted with datasets that consist of uniformly dispersed symbols, like random noise or some types of multimedia files, its performance may suffer.

CONCLUSION

These findings illustrate the limitations and benefits of the various compression methods, with Huffman appearing as the more efficient and effective text compression option in our studies. Huffman coding outperformed previous approaches by obtaining greater compression ratios, especially for datasets containing largely text-based content. Because of the algorithm's capacity to apply shorter codes to commonly appearing symbols in the data, file sizes were significantly reduced, which makes it appropriate for textual material such as texts, documents, and source code. Finally, the Burrows-Wheeler Transform (BWT) shown distinct advantages in certain cases, particularly when combined with other compression algorithms. BWT successfully reorganized the data to allow for higher compression ratios for a variety of data types, including DNA sequences or textual material with repeating structures. However, its processing overhead and the need for additional techniques, such as Move-to-Front (MTF) or Run-Length Encoding, can result in complexity that, in some situations, outweighs its benefits. Moving from one-bit-at-a-time renormalizations to those which group bits together into bite-sized organisations or larger provides an important speed boost. To accommodate a larger variety of period durations, byte-based renormalizations need arithmetic operations with adequate accuracy (e.g., a minimum of 16 or 32 bits). Instead of approximations, such accuracy can be handled efficiently by native CPU processes. Multiplications have gotten significantly faster, with little influence on coding speed, even for stationary binary coders. Although binary coders perform coding operations quickly, their data capacity is usually restricted to one bit per cycle at most. It is best to use algorithms that code symbols using larger letters to attain the fastest coding speeds, since they can give much higher throughputs and expected outcomes.

Future Scope

The study areas listed below offer intriguing prospects for additional investigation in the realm of compression algorithms. In addition to the present Shannon-Fano, Huffman, & Burrows-Wheeler Transform (BWT) approaches, future research could examine adding new compression algorithms including Run-Length Encoding (RLE), Move-to-Front (MTF), or LZ77. Furthermore, examining the execution of a mixture of BWT, Huffman, & Shannon-Fano algorithms on various file types such as photos, audio, and video can broaden the field of compression techniques beyond text data. In the years to come, a system based on sensors could be potentially created to automatically recognize the file type and then select the best possible compression technique for that specific file. With the growing volume of data that is transmitted and stored on a daily basis, the necessity for more effective compression methods becomes critical. Efforts can be put towards improving compression ratios in current systems, especially when source entropy is low (e.g., less than 3 bits/symbol). Exploring search algorithms that rely exclusively on multiplications and optimizing the search sequence can result in improved compression efficiency in such circumstances. Emphasizing these research areas will help to develop compression technologies and allow for more efficient processing of massive datasets in a variety of application fields.

REFERENCES

Ahmad, T., Madonski, R., Zhang, D., Huang, C., & Mujeeb, A. (2022). Data-driven probabilistic machine learning in sustainable smart energy/smart energy systems: Key developments, challenges, and future research opportunities in the context of smart grid paradigm. *Renewable and Sustainable Energy Reviews*, *160*, 112128.

Al-Qurabat, M., & Kadhum, A. (2021). A lightweight Huffman-based differential encoding lossless compression technique in IoT for smart agriculture. *International Journal of Computing and Digital System*.

Anaspure, M. S. (2022). *Automation in cloud environment using cloud services and python script* (Doctoral dissertation, Dublin, National College of Ireland).

Ashila, M. R., Atikah, N., Rachmawanto, E. H., & Sari, C. A. (2019, October). Hybrid AES-Huffman Coding for Secure Lossless Transmission. In *2019 Fourth International Conference on Informatics and Computing (ICIC)* (pp. 1-5). IEEE.

Begum, M. B., Deepa, N., Uddin, M., Kaluri, R., Abdelhaq, M., & Alsaqour, R. (2023). An efficient and secure compression technique for data protection using burrows-wheeler transform algorithm. *Heliyon*.

Bello, H. B. (2020). *Bidirectional Search in a String Using Wavelet Matrix and Burrows Wheeler Transform* (Doctoral dissertation, AUST).

Biagetti, G., Crippa, P., Falaschetti, L., Mansour, A., & Turchetti, C. (2021). Energy and performance analysis of lossless compression algorithms for wireless emg sensors. *Sensors*, *21*(15), 5160.

Erdal, E., & Ergüzen, A. (2019). An efficient encoding algorithm using local path on huffman encoding algorithm for compression. *Applied Sciences*, *9*(4), 782.

Fitzgerald, R. M. (2020). WAKING TO NORMAL: Examining Archival Appraisal in Data-driven Society.

Giuliani, S., Inenaga, S., Lipták, Z., Prezza, N., Sciortino, M., & Toffanello, A. (2021). Novel results on the number of runs of the burrows-wheeler-transform. In *SOFSEM 2021: Theory and Practice of Computer Science: 47th International Conference on Current Trends in Theory and Practice of Computer Science, SOFSEM 2021, Bolzano-Bozen, Italy, January 25–29, 2021, Proceedings 47* (pp. 249-262). Springer International Publishing.

Grewal, K. S. (2021). *Lossless Data Compression by Representing Data as a Solution to the Diophantine Equations* (Doctoral dissertation, Arizona State University).

Gudodagi, R., Akash, K. T., & Ahmed, M. R. (2023, February). Deep Learning Algorithms for Secure and Efficient Compression of Genomic Sequence Data. In 2023 IEEE 3rd International Conference on Technology, Engineering, Management for Societal impact using Marketing, Entrepreneurship and Talent (TEMSMET) (pp. 1-7). IEEE.

Gui, S., Wang, H., Yang, H., Yu, C., Wang, Z., & Liu, J. (2019). Model compression with adversarial robustness: A unified optimization framework. Advances in Neural Information Processing Systems, 32.

Hidayat, T., Zakaria, M. H., & Pee, A. N. C. (2020, November). Survey of performance measurement indicators for lossless compression technique based on the objectives. In *2020 3rd International Conference on Information and Communications Technology (ICOIACT)* (pp. 170-175). IEEE.

Karim, A. Z., Miah, M. S., Al Mahmud, M. A., & Rahman, M. T. (2021, September). Image Compression using Huffman Coding Scheme with Partial/Piecewise Color Selection.

In *2021 IEEE 4th International Conference on Computing, Power and Communication Technologies (GUCON)* (pp. 1-6). IEEE.

Kumar, S., Agarwal, S., & Ranvijay. (2019). Fast and memory efficient approach for mapping NGS reads to a reference genome. *Journal of Bioinformatics and Computational Biology*, *17*(02), 1950008.

Kumari, P., & Saini, M. (2022). Anomaly Detection in Audio With Concept Drift Using Dynamic Huffman Coding. *IEEE Sensors Journal*, *22*(17), 17126-17138.

Lawal, T. D., Olatunbosun, L. O., & Gbolagade, K. A. (2021). An Improve Shannon Fano Data Compression Algorithm using Residue Number System. *Commun. Appl. Electron.*, *7*(35), 19-25.

Liu, X., An, P., Chen, Y., & Huang, X. (2022). An improved lossless image compression algorithm based on Huffman coding. *Multimedia Tools and Applications*, *81*(4), 4781-4795.

Long, F., Wang, L., Cai, W., Lesnik, K., & Liu, H. (2021). Predicting the performance of anaerobic digestion using machine learning algorithms and genomic data. *Water Research*, *199*, 117182.

Ma, C., Liu, D., Peng, X., Li, L., & Wu, F. (2019). Convolutional neural network-based arithmetic coding for HEVC intra-predicted residues. *IEEE Transactions on Circuits and Systems for Video Technology*, *30*(7), 1901-1916.

Mallik, S., & Zhao, Z. (2020). Graph-and rule-based learning algorithms: a comprehensive review of their applications for cancer type classification and prognosis using genomic data. *Briefings in bioinformatics*, *21*(2), 368-394.

Mentzer, F., Gool, L. V., & Tschannen, M. (2020). Learning better lossless compression using lossy compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 6638-6647).

Moffat, A. (2019). Huffman coding. *ACM Computing Surveys (CSUR)*, *52*(4), 1-35.

Mohammadi, H., Ghaderzadeh, A., & Sheikh Ahmadi, A. (2022). A novel hybrid medical data compression using huffman coding and LZW in IoT. *IETE Journal of Research*, 1-15.

Nassra, I., & Capella, J. V. (2023). Data Compression Techniques in IoT-enabled Wireless Body Sensor Networks: A Systematic Literature Review and Research Trends for QoS Improvement. *Internet of Things*, 100806.

Nosratian, S., Moradkhani, M., & Tavakoli, M. B. (2021). Hybrid data compression using fuzzy logic and Huffman coding in secure IOT. *Iranian Journal of Fuzzy Systems*, *18*(1), 101-116.

Oswald, C., & Sivaselvan, B. (2023). Smart Multimedia Compressor—Intelligent Algorithms for Text and Image Compression. *The Computer Journal*, *66*(2), 463-478.

Otair, M., Abualigah, L., & Qawaqzeh, M. K. (2022). Improved near-lossless technique using the Huffman coding for enhancing the quality of image compression. *Multimedia Tools* and Applications, 81(20), 28509-28529.

Rahman, M. A., & Hamada, M. (2019). Lossless image compression techniques: A state-of-the-art survey. *Symmetry*, *11*(10), 1274.

Rahman, M. A., & Hamada, M. (2020). Burrows–wheeler transform based lossless text compression using keys and Huffman coding. *Symmetry*, *12*(10), 1654.

Russell, M., & Wang, P. (2022). Physics-informed deep learning for signal compression and reconstruction of big data in industrial condition monitoring. *Mechanical Systems and Signal Processing*, *168*, 108709.

SANDHU, S. (2021). LOSSLESS DATA COMPRESSION: AN OVERVIEW.

Sarangi, S. (2022). *Hardware Architectures for Lossless Compression*. eScholarship, University of California.

Seeliger, R., Müller, C., & Arbanowski, S. (2022, November). Green streaming through utilization of AI-based content aware encoding. In *2022 IEEE International Conference on Internet of Things and Intelligence Systems (IoTaIS)* (pp. 43-49). IEEE.

Shah, T. J., & Banday, M. T. (2020). A review of contemporary image compression techniques and standards. *Examining Fractal Image Processing and Analysis*, 121-157.

Sharma, N., & Batra, U. (2020). Performance analysis of compression algorithms for information security: A Review. *EAI Endorsed Transactions on Scalable Information Systems*, *7*(27).

Townsend, J. (2021). Lossless compression with latent variable models. *arXiv preprint* arXiv:2104.10544.

Umbaugh, S. E. (2022). *Digital Image Processing and Analysis: Digital Image Enhancement, Restoration and Compression*. Crc Press.

Williams, D. M. (2022). *Performance Overhead of Lossless Data Compression and Decompression Algorithms: A Qualitative Fundamental Research Study* (Doctoral dissertation, Northcentral University).

Wright, J., & Ma, Y. (2022). *High-dimensional data analysis with low-dimensional models: Principles, computation, and applications*. Cambridge University Press.

Zhang, B., Tian, J., Di, S., Yu, X., Swany, M., Tao, D., & Cappello, F. (2023, June). GPULZ: Optimizing LZSS Lossless Compression for Multi-byte Data on Modern GPUs. In *Proceedings of the 37th International Conference on Supercomputing* (pp. 348-359).