Efficiency Evaluation Between Arithmetic and Huffman Coding for Data Compressions

Zahed Shikilkar [1], Ganesh Shinde [2], Girish Ingale [3], Tejas Mukkawar [4], Dipti Pandit [5]

Student, Electronics and Telecommunications, Vishwakarma Institute of Information Technology, Pune, India [1 2 3 4]
Assistant Professor, Electronics and Telecommunications, Vishwakarma Institute of Information Technology Pune, India [5]

***Abstract*—The ever increasing generation of digital data and its applications in multimedia, cloud computing, communication systems etc require efficient methods of data compression for storage and transmission purposes. Conventional data compression methods are applied and used with the specific goal to produce a minimal file size while at the same time preserving data integrity hence, lossless compression is very vital in instances whereby the original data must be reconstructed to the precise representation. This work examines two widely used lossless compression methods: Huffman coding and Arithmetic coding. The performance of both techniques is compared using the following parameters: the length of the compressed data, entropy, redundancy in bits, and the rate of compression on a text data set and an audio file. This paper verifies the influence of the proposed approaches in text and audio compression experiments through Arithmetic coding and Huffman coding, respectively; the later yields a better result due to its binary characteristic. The results imply that the choice should be made focusing on the peculiarities of the data before defining the compression method.**

***Index Terms*—entropy, redundancy, arithmetic coding, Huffman coding, compression ratio, and audio**

I. Introduction

As the availability of social media, cloud storage multimedia, and IoT devices has increased exponentially the management of large amount of data has become important [1] [2]. However, current and future advances in computers and communication technology are rapidly generating a massive amount of data and we require compression techniques where the amount of data to be stored and transmitted is greatly minimized but the essence of the information is retained. There are several performances for data compression for instances in transferring multimedia information, transmitting data on networks, and minimizing storage space [3].With the use of lossless data compression algorithms, it is possible to precisely rebuild the original data from the compressed data without any information loss. For applications like text files, medical photographs, or legal papers where data correctness is crucial, they are therefore indispensable [1].

Different techniques are used in data compression to minimize file sizes without compromising data integrity. Arithmetic coding and Huffman coding are two popular lossless compression methods that assign codes based on symbol frequency. When applied to datasets with known symbol probabilities, Huffman coding—an entropy-based method—assigns variable-length codes to symbols. However, when symbol probabilities are extremely skewed, arithmetic coding frequently achieves greater compression ratios since it encodes the entire message as a single fractional integer. Arithmetic coding is more versatile than Huffman coding, but it requires more computing power.

Run-Length Encoding (RLE) and Lempel-Ziv-Welch (LZW) are two further compression techniques that are important for data compression. Specifically designed for text compression, dictionary-based latent zone wording (LZW) efficiently handles repetitive data by substituting dictionary references for sequences of recurrent data patterns. However, in situations where symbol distributions are erratic, Huffman and arithmetic coding usually perform better than LZW. Compared to Huffman or arithmetic coding, RLE is less flexible as it only works with extremely repetitive material. It is a simpler method of data compression that involves encoding repeated sequences [2]–[4].

Conversely, lossy compression methods—like MP3 for audio—reduce file sizes by removing less noticeable information, sacrificing some fidelity in exchange for a substantial size reduction [5], [6]. Huffman coding and Arithmetic coding preserve data integrity, making them more appropriate for applications that require lossless compression, even though lossy approaches are useful in multimedia applications.

II. Background

Lossless compression techniques are essential for maintaining data accuracy while lowering file sizes for effective transmission and storage in an era where storage and bandwidth needs are rising. Robust compression algorithms are essential for the efficient management of data quantities in applications such as cloud storage optimization, data archival, and real-time multimedia streaming [7].

In situations where data integrity cannot be jeopardized, lossless compression techniques guarantee that the original data may be precisely restored from the compressed form. These techniques are not the same as lossy compression methods, like JPEG for images and MP3 for audio, which reduce file size by sacrificing some fidelity in order to attain better compression ratios [5], [6]. Although lossless compression is essential for applications involving text files, medical imaging, legal papers, and other sectors where precise reconstruction is crucial, lossy compression is suitable for the transfer of multimedia information, where small quality losses are acceptable.

Arithmetic coding and Huffman coding are two of the most used algorithms among lossless compression approaches. David Huffman developed Huffman coding in 1952, and it is a fundamental method where symbols are given variable-length codes according to how frequently they occur in the dataset. Huffman coding is quite effective for datasets with known and consistent symbol distributions, but it can lose some of its optimality for severely skewed or irregular symbol probabilities [7], [8]. When symbol probabilities are not uniform, arithmetic coding—which reduces the entire message to a single fractional number—offers a more efficient method of compression. Arithmetic coding requires more resources than Huffman coding because of its higher computational complexity, which is the price paid for its increased efficiency [2].

Apart from Huffman and Arithmetic coding, alternative techniques for compression include Lempel-Ziv-Welch (LZW) and

Run-Length Encoding (RLE) [4], which are lossless methods. LZW is a dictionary-based technique that works well with highly redundant text input by substituting references to a dictionary of patterns for sequences of symbols. Huffman and arithmetic coding often perform better; however, LZW has trouble with datasets with irregular symbol distributions. RLE is a more straightforward method that compresses data containing repetitive sequences, like some kinds of images, very effectively. It does this by encoding consecutive runs of repeated symbols. RLE can't, however, be applied to more complicated datasets with unpredictable redundancy, such as text or audio [2], [3].

The efficiency of data transmission and storage has greatly increased with the invention of various compression techniques. The best compression technique, however, is frequently determined by the particulars of the data. For example, arithmetic coding might be more effective for text compression when symbol frequencies are predictable, but Huffman coding works better for audio data with more unpredictable distributions. While Huffman and Arithmetic Coding offer greater flexibility for a wider range of datasets, LZW and RLE are good substitutes in some usage scenarios [2], [3].

III. Essential Evaluation Metrics

In comparing the two methods, the following essential evaluation metrics were used:

1. Entropy (H(X)): Entropy measures the theoretical limit of average code length and is calculated using the formula [9],

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log_2 p(x_i) \quad (1)$$

where $p(x_i)$ *is the probability of symbol* x_i *and* n is the number of unique symbols.

2. Average Code Length (L): The average code length is determined by the formula [9],

$$L = i = 1^{n} p(x_i) * l(x_i) \quad (2)$$

where $l(x_i)$ *represents the length of the code for symbol* x_i.

3. Redundancy (R): Redundancy reflects the variance between the actual code length and the theoretical entropy [9],

$$calculatedasR = L - H(X). \quad (3)$$

4. Compression Ratio (CR): The compression ratio is defined as the ratio of the original data size to the compressed size,expressed as [9]

$$expressedasCR = OriginalSize/CompressedSize. \quad (4)$$

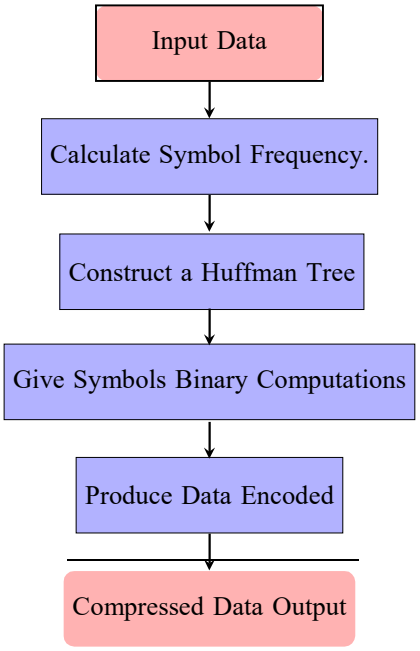
IV. Methodology

A. Huffman Coding

1) Huffman Coding Algorithm:

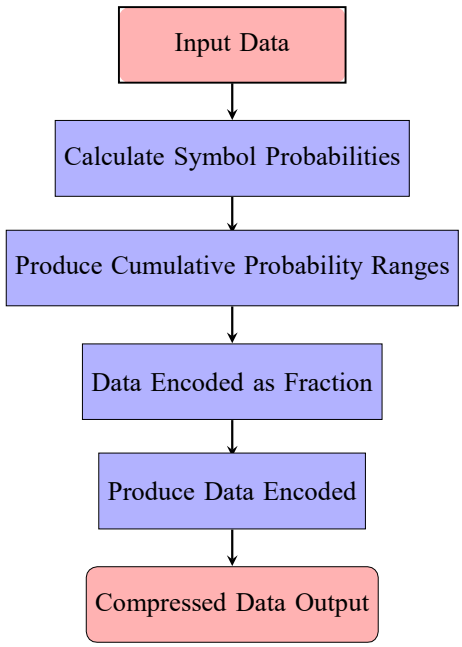
Algorithm 1 Huffman Coding

1: Calculate Frequency
2: Make a priority queue and put all the symbols together with their frequencies.
3: **while** there is more than one node in the queue **do**
4: Drop the two symbols with the least frequency count.
5: Create a new internal node with these two nodes as child nodes and set its frequency as the sum of their frequencies.
6: Put the new node back into the queue to be compared against other symbols.
7: **end while**
8: When only one node remains, it forms the root of the Huffman Tree.
9: Assign Codes
10: **for** each symbol in the Huffman Tree **do**
11: Walk through the tree, assigning '0' along the left edges and '1' along the right edges.
12: Assign binary codes to each symbol based on the path from the root to the symbol.
13: **end for**
14: Generate Encoded Data
15: To get the compressed output, replace each symbol in the original dataset with its corresponding Huffman code.



Algorithm 2 Arithmetic Coding

1: Calculate Probabilities
2: Calculate the probability of each symbol in the data set..
3: Create Cumulative Probabilities.
4: build a cumulative probability table for the symbols. .
5: Set Initial Interval
6: Define the initial interval as [0, 1).
7: Encode the Message
8: **for** each symbol in the dataset **do**
the interval according to the cumulative probability.Calculate the new interval boundaries using the current symbol's cumulative probability.
9:10: **end for**
11: Output the Final Value any value from the final interval as the encoded output..



V. Results

A. Text Dataset Results

Arithmetic Coding:

- Compressed Size: 10 bits
- Compression ratio: 0.1042
- Active Length: 1.2784 bits/symbol
- Entropy: 1.2783 bits/symbol
- Redundancy: 0

. **Huffman Coding:**

- Compressed Size: 17 bits
- Compression ratio: 0.1771
- Active Length: 1.4167 bits/symbol
- Entropy: 1.2783 bits/symbol
- Redundancy: 0.1384

B. Text Dataset Results

This could clearly be observed from the results captured in the results table indicating the differences between Huffman coding and Arithmetic coding. Huffman has been compressed to possess a size of 17 bits while Arithmetic has been compressed to possess the smallest size of 10 bits. This means that Arithmetic coding definitely brings down the data size to a lower level of more effectiveness in the aspects of storage and transmission.

When analysing the compression ratio, Huffman coding provides a ratio of 0.1771 with the compressed data being approximately 17.71 percent in size to the original. For instance Arithmetic coding yields a slightly improved compression rate of 0.1042 thus occupying only 10.42 percent of the original space. They both share the same entropy of 1.2783 bits per symbol, that is, the performance achieved is the same for each of the symbols.

However, Huffman coding has some inefficiencies, for example a redundancy of 0.1384 while Arithmetic coding has zero redundancy which simply means that every bit in the image has been used without wastage. Even the number average is the number of bits assigned to encoded symbols and is observed that the Huffman coding is the average of about 1.4167 bits per symbol while Arithmetic coding has a much lower average of only 1.2783 bits per symbol which is quite optimal. Based in these results, one could conclude that in general, Arithmetic coding is preferred for lossless data compression since it compresses data to a lower size, hence using smaller space.

TABLE I
Comparison between Huffman and Arithmetic Coding

Parameter	Huffman Coding	Arithmetic Coding
Compressed Size	17 bits	10 bits
Compression Ratio	0.1771	0.1042
Entropy	1.2783 bits/symbol	1.2783 bits/symbol
Redundancy	0.1384	0
Average Length	1.4167 bits/symbol	1.2783 bits/symbol

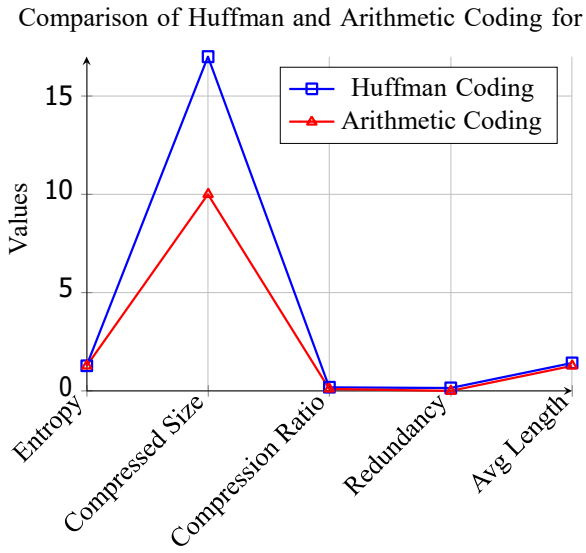


Fig. 1. Comparison of Huffman and Arithmetic Coding based on various parameters.

C. Audio Dataset Results

The following results were obtained for the audio file:

Huffman Coding:

- Entropy: 14.37 bits/symbol
- Redundancy: 0.04 bits/symbol
- Compression Ratio: 1.11
- Average Code Length : 14.40 bits

Arithmetic Coding:

- Entropy: 14.37 bits/symbol
- Redundancy: 0.0 bits/symbol
- Compression Ratio: 1.14
- Average Code Length : 14.37 bits

D. Audio Dataset Results

For the audio dataset, the results are shown in Table II Thus, the table below compares the Huffman coding with the Arithmetic coding on the basis of certain important parameters. The entropy for both methods turned out to be identical and equaled 14.37 bits, thus allowing providing a similar amount of information for each symbol in the given data set. This is not bad news because both techniques are useful for capturing such data and they both appear to perform reasonably well.

We are also able to determine that the Huffman coding takes an average of approximately 14.40 bits per symbol while Arithmetic coding takes a slightly lower average of about 14.37 bits per symbol on average. This means that Arithmetic coding can use fewer bits to represent the same information, which is always an advantage when it comes to the size of files that need to be stored.

Third, the table below presents the compression ratio asserting the effectiveness of each method in minimizing the size of the data. While comparing the Huffman with the

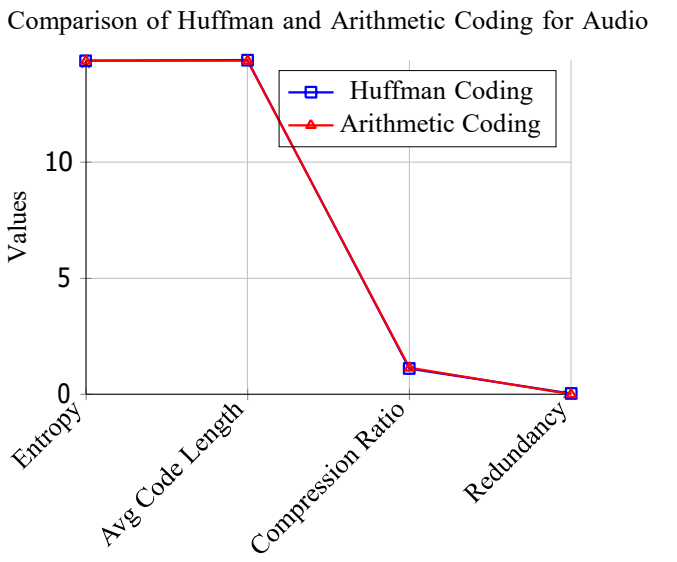


Fig. 2. Comparison of Huffman and Arithmetic Coding based on various parameters

Arithmetic coding, the respective ratio of the frequency of code words is 1.11 while that of the Arithmetic coding is slightly lower than the Huffman, with a frequency of ratio 1.14. This mean that the method is able to better compress the data based on the higher compression ratio recorded. However, Arithmetic coding performs a better job of converting the size of the data as compared to Huffman coding.Last but not the least, the implication of redundancy metric is to determine the manner which each method employs bits. Huffman coding has a redundancy of 0.03 bits, so the amount of overhead in Huffman coding is extremely small. On the other hand Arithmetic coding has zero redundancy that is 0.0 bits, implying that all the bits are used appropriately and no extra bit is used.

Therefore, we can infer from Table 1 that, in general, in comparison to the Huffman coding, Arithmetic coding is a better compression technique in sense of average code length as well as the ration achieved and has advantage over Huffman both in affecting space also in regard with redundancy. All these therefore make Arithmetic coding particularly desirable for applications that seek to achieve the highest levels of data compression.

TABLE II
COMPARISON BETWEEN HUFFMAN CODING AND ARITHMETIC CODING

Metric	Huffman Coding	Arithmetic Coding
Entropy	14.37 bits	14.37 bits
Average Code Length	14.40 bits	14.37 bits
Compression Ratio	1.11	1.14
Redundancy	0.03 bits	0.0 bits

VI. ANALYSIS

The results presented for both text and audio show how Huffman coding and Arithmetic coding works and its performance. Looking at the text dataset, the Arithmetic coding is compressed with only 10 bits, while Huffman coding has compressed the data to 17 bits, hence it is clear that Arithmetic coding is much more efficient in minimizing data size through use of fewer bits to represent the same data. The efficiency of both methods is estimated to be equal to 1.2783 bits/symbol, thus, both methods provide the same amount of information per symbol. Nevertheless, Huffman coding has 0.1384 bits of redundancy, which speaks to the fact that there is some inefficiency to be found in Huffman's encoding, while Arithmetic coding has none, or 0.0 bits, meaning it is true to form and uses all of the bits without any extra or spare. This has clearly shown that Arithmetic coding has a better compression ratio for the text dataset than any of the two mentioned methods.

The encoding techniques show that, for the audio dataset, both techniques have an entropy of 14.37 bits per symbol which powerfully suggest that they encode the same amount of information. Comparing with the Huffman coding and Arithmetic coding, this is how the average code length looks like, Average code length = 14.40 bits symbol for Huffman coding and slight better with 14.37 bits symbol for Arithmetic coding. As indicated regarding the compression ratio, the Huffman coding has a value of 1.14, which is slightly higher than that recorded in Arithmetic coding of 1.11. Huffman coding may be even more efficient in this sense, reminding that this method is generally thought to yield better results in reducing the size of files. Moreover, analysis based on the redundancy metric shows that redundancy of Huffman coding is 0.03 bits while redundancy of Arithmetic coding is 0.00, which suggest the efficiency of Arithmetic coding.

Taken together, lessons from both datasets underscore the merits and demerits of each form of coding. Based on the comparison of the compressed text-dataset results, Arithmetic coding seems to require lesser bits to compress and has less redundancy than Huffman coding method to be preferred for data compression. Analyzing the results of the audio dataset it is possible to see that although both methods are quite similar by entropy and redundancy metrics, Huffman coding is slightly better by the rate of compression ratio. But Arithmetic coding is superior in average code length and has no matter of fact zero redundancy. This implies that although Huffman coding can work, especially for Huffman coding audio data, Arithmetic coding often outperforms other methods, especially if reducing data size and improving efficiency is tenable. The decision of which method has to be used has to take into account properties and additional need of the dataset to be compressed.

VII. CONCLUSION

Altogether, the analysis of Huffman coding and Arithmetic coding enables understanding of the possible directions in data compression development. In realisation of the different data set scenarios, arithmetic coding appears to offer the shortest length of the compressed data set with the least redundant and as such is the better data compression model. However, as clearly seen, these two methods suggest the same entropy, and the average code length and redundancy make Arithmetic coding more beneficial.

For recent coded entropy in audio dataset, both techniques have approximately equal entropy, but the compression ratio was found little higher in case of Huffman coding than that of Arithmetic coding. However Arithmetic coding has slightly lower mean freedom ratio and zero redundancy to back up its efficiency.

So, both Huffman coding and Arithmetic coding are ways of data compression though depending with the nature and needs of a given data set, Huffman or Arithmetic coding should be preferred. Arithmetic coding is more appropriate in most cases when the size of the storage file and the efficiency of the encoding are significant; Huffman is beneficial when the compression ratio is essential, for example, in the case of audio data. Subsequent studies may extend the presented approach toward

novel datasets and additional practical scenarios to determine specific optimal methods of lossless data compression.

REFERENCES

[1] R. Weizheng, W. Haobo, X. Lianming, and C. Yansong, "Research on a quasi-lossless compression algorithm based on huffman coding," in *Proceedings 2011 International Conference on Transportation, Mechanical, and Electrical Engineering (TMEE)*, pp. 1729–1732, Dec 2011.

[2] L. Barua, P. K. Dhar, L. Alam, and I. Echizen, "Bangla text compression based on modified lempel-ziv-welch algorithm," in *2017 International Conference on Electrical, Computer and Communication Engineering (ECCE)*, pp. 855–859, Feb 2017.

[3] S. Bhattacharjee, S. K. Choudhury, S. Das, and A. Pramanik, "Dpcm block-based compressed sensing with frequency domain filtering and lempel-ziv-welch compression," in *2015 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pp. 1244–1249, Aug 2015.

[4] A. Birajdar, H. Agarwal, M. Bolia, and V. Gupte, "Image compression using run length encoding and its optimisation," in *2019 Global Conference for Advancement in Technology (GCAT)*, pp. 1–6, Oct 2019.

[5] H.-A. Pham, V.-H. Bui, and A.-V. Dinh-Duc, "An adaptive huffman decoding algorithm for mp3 decoder," in *2010 Fifth IEEE International Symposium on Electronic Design, Test Applications*, pp. 153–157, Jan 2010.

[6] C.-W. Huang, J. J. Thiagarajan, A. Spanias, and C. Pattichis, "A java-dsp interface for analysis of the mp3 algorithm," in *2011 Digital Signal Processing and Signal Processing Education Meeting (DSP/SPE)*, pp. 168–173, Jan 2011.

[7] P. Mbewe and S. D. Asare, "Analysis and comparison of adaptive huffman coding and arithmetic coding algorithms," in *2017 13th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)*, pp. 178–185, July 2017.

[8] S. P. Venkatesan, S. Srividhya, N. Saikumar, and G. Manikandan, "Generating strong keys using modified huffman tree approach," in *2016 International Conference on Circuit, Power and Computing Technologies (ICCPCT)*, pp. 1–4, March 2016.

[9] R. Arshad, A. Saleem, and D. Khan, "Performance comparison of huffman coding and double huffman coding," in *2016 Sixth International Conference on Innovative Computing Technology (INTECH)*, pp. 361–364, Aug 2016.