Efficient DNA Sequence Compression: A Comparative Study of Huffman and Arithmetic Coding with Data Structures

Atharva Vishwas Deshpande
Dept. of Electronics & Telecommunication
Vishwakarma Institute of Information Technology
Pune, India
atharva.22311679@viit.ac.in

Samiksha Shailesh Nalawade
Dept. of Electronics & Telecommunication
Vishwakarma Institute of Information Technology
Pune, India
samiksha.22311696@viit.ac.in

Revati Tushar Aute
Dept. of Electronics & Telecommunication
Vishwakarma Institute of Information Technology
Pune, India
revati.22311517@viit.ac.in

Manas Girish Kulkarni
Dept. of Electronics & Telecommunication
Vishwakarma Institute of Information Technology
Pune, India
manas.22311608@viit.ac.in

Rajlakshami Nilesh Desai
Dept. of Electronics & Telecommunication
Vishwakarma Institute of Information Technology
Pune, India
rajlakshami.22311732 @viit.ac.in

Dr. Minal Samir Deshmukh
Asst. Professor, Dept. of Electronics & Telecommunication
Vishwakarma Institute of Information Technology
Pune, India
minal.deshmukh@viit.ac.in

Abstract*— *The rapid growth of genomic data poses difficulties in storing and transferring information efficiently. This study offers a comparison of two methods of compression- Huffman Coding and Arithmetic Coding. When used on DNA sequences. Huffman Coding uses a Binary Search Tree (BST) approach for optimal prefix codes whereas Arithmetic Coding compresses data based on probability ranges. Results from experiments indicate that both techniques decrease memory usage, but differ in compression ratio, memory footprint, and computational complexity. Huffman Coding achieves a compression ratio of 0.33 and faster encoding, while Arithmetic Coding reaches 0.051, offering finer compression granularity. This comparison emphasizes the balance between compression efficiency and processing speed with implications, for bioinformatics applications.

Keywords— Huffman Encoding, Arithmetic Encoding, DNA compression

I. INTRODUCTION

The swift progress in next-generation sequencing (NGS) technologies have resulted in an unprecedented surge in genomic data generation. With sequencing becoming more accessible, handling the amount of DNA sequence data presents hurdles for bioinformatics experts and storage systems. Efficient compression techniques are essential for addressing increasing storage demands and transmission bottlenecks.

Genomic sequences, consisting of four nucleotide bases—adenine (A), cytosine (C), guanine (G), and thymine (T)—exhibit repetitive patterns that can be leveraged for data compression. However, traditional algorithms like Gzip or Bzip2 are not optimized for DNA sequences, leading to suboptimal performance. Lossless techniques, such as Huffman Coding and Arithmetic Coding, utilize frequency and probability-based approaches, presenting promising alternatives.

Efficient storage and manipulation of large biological datasets, particularly DNA sequences, remain significant challenges in bioinformatics.[1]introduces EIBDNASCA, which optimizes an index file for non-repetitive bases and a work file for repetitive bases, achieving a compression ratio of 1.23 bpb with a gain of 84.52%. Advanced compression techniques like LZMA are vital for optimizing storage and transmission, as shown in [7]where LZMA's effectiveness in reducing DNA sequence size while maintaining data integrity is highlighted.

This study presents a comparative analysis of Huffman and Arithmetic Coding for compressing DNA sequences. The Binary Search Tree (BST)-based Huffman implementation constructs an optimal prefix-free binary tree using nucleotide frequency data, while Arithmetic Coding assigns probability intervals to each nucleotide. Performance evaluations across various genomic datasets measure compression ratio, memory efficiency, and computational complexity. Results indicate that Huffman Coding offers faster encoding times, whereas Arithmetic Coding achieves finer compression ratios, especially in datasets with non-uniform nucleotide distributions.

This research enhances bioinformatics by offering an examination of these compression methods and pointing out their advantages and drawbacks in enhancing the storage and transfer of genomic data in extensive projects. The structure of this paper is as follows: Section II provides a literature review, highlighting developments in DNA compression techniques. Section III explains the proposed system explaining how DNA compression can be achieved using both Huffman and Arithmetic Encoding techniques. Section IV shows the memory analysis and compression efficiency. Section V discusses the performance and accuracy of the algorithm by decoding the DNA back to its original form. Section VI explores the application of both the techniques based on their performance and accuracy, and lastly Section VII discusses limitations and future scope for further advancement. Section VIII concludes the paper.

II. LITERATURE SURVEY

Gurunathan et al. (2021) [1] introduced an enhanced index-based DNA sequence compression algorithm. The paper presents a novel approach to compressing DNA sequences by creating a structured index, which facilitates efficient storage and retrieval of genetic information. This method not only reduces the size of DNA data but also accelerates the retrieval process, making it particularly useful for bioinformatics applications. The results demonstrated that this approach outperforms traditional compression techniques in terms of both compression ratio and retrieval speed. Ali Hassan et al. (2024) [2]presented a model that combines the N-gram language model with arithmetic coding for better data compression. An application of utmost interest here is in high storage efficiency applications. It discusses traditional techniques such as Huffman and LZW, pointing out the limitations involving individual character encoding. The proposed ArthNgram model makes use of word sequences improving compression ratios with better performance in compression speed and storage reduction than conventional methods. The simulated results of the proposed model are validated by balancing the compression ratio with the processing time suitable in applications where data is large or in IoT. Babita Kumari et al. (2023)[2]. Their paper enhances the Adaptive Huffman algorithm for text compression by using text clustering and multiple character modification. This approach builds on limitations in prior Huffman techniques with large repetitive patterns by grouping similar text and modifying clusters to reduce redundancy, ultimately achieving higher compression ratios and improved efficiency. A. Hussein et al. (2023) [4]explored a novel approach to data hiding in RGB images using a combination of genetic algorithms and Huffman coding. Although primarily focused on image compression, the study emphasizes the flexibility of Huffman coding and its potential for various types of data, including DNA sequences. This approach demonstrates the adaptability of Huffman coding and genetic algorithms in improving data handling for DNA compression, highlighting the method's versatility beyond its traditional applications. M. Alosta et al. (2022) [5] have discussed a lossless compression technique specifically designed for multiple DNA sequences. By utilizing Huffman encoding along with other strategies, the authors ensure that the original data remains intact while achieving high compression rates. The study emphasizes the importance of effective compression techniques for diverse genomic datasets and demonstrates that their method is both efficient and reliable. Mehedi Hasan Sarkar et al. (2022) [6] have introduced a novel method that combines machine learning and arithmetic encoding techniques to effectively compress genetic sequences. This approach identifies patterns in genomic data, resulting in better compression ratios than traditional methods. The authors provide experimental evidence demonstrating improved efficiency and data integrity, showcasing the promise of integrating machine learning with compression techniques in bioinformatics. D. Gomathi et al. (2021) [7] combined two compression techniques—Run-Length Encoding (RLE) and Huffman encoding—to compress DNA sequences. RLE reduces redundancy by replacing consecutive identical nucleotides with a single value, while Huffman encoding optimizes the binary code based on nucleotide frequency. Their study demonstrated that this hybrid approach achieves superior compression rates compared to using either method individually, effectively addressing the repetitive patterns commonly found in DNA sequences. M. Ganapathy et al. (2021) [8] have presented an IT reduction-based compression algorithm for DNA sequences, focusing on reducing redundancy in genomic data. The method leverages information theory to efficiently compress DNA sequences while maintaining data accuracy. The study demonstrates promising results, highlighting its potential for enhancing storage efficiency in genomic data management. This contribution is significant for addressing the growing challenges in the field of bioinformatics, particularly with the increasing volume of DNA sequence data. J. Zhao et al. (2021) [9] have proposed enhancements to traditional Huffman coding to address the challenges of compressing large-scale DNA sequences. The study integrates data pre-processing steps that group similar regions of the genome, allowing for more efficient encoding using Huffman trees. Their experimental results demonstrated improved performance in terms of both compression ratio and speed when compared to standard Huffman methods, offering a more effective solution for genomic data compression. P. Mishra et al. (2021)[10]have introduced a novel approach for compressing DNA sequences using a Minimum Variance Huffman Tree (MVHT). The study optimized the traditional Huffman coding method by minimizing the variance in code length, which led to improved compression efficiency. The algorithm effectively addressed the unique nucleotide frequency distributions in DNA sequences. contexts Experimental results demonstrated that MVHT outperformed standard Huffman coding by achieving better compression ratios while preserving the original data's integrity. This research contributes to more efficient storage solutions in the field of bioinformatics. Pulova-Mihaylova et al. (2020) [11] developed a web-based system for the compression of sequencing data, addressing the challenges posed by the increasing volume of genomic data generated by modern sequencing technologies. The study introduces an Optimized Algorithm that enhances traditional methods like Huffman and Shannon-Fano by implementing uniform coding, which improves both compression efficiency and error resilience. Results indicate that this system not only accelerates data processing but also integrates seamlessly with existing bioinformatics platforms, facilitating better management of large-scale genomic datasets. A. Jahaan et al. (2017) [12] have discussed the increasing need for efficient DNA data storage due to the exponential growth of genetic research. The paper emphasizes the requirement for lossless compression techniques, evaluating several existing algorithms like Biocompress and DNACompress. The authors highlight the importance of identifying and encoding repetitive sequences to enhance compression efficiency. Machhi and Patel (2016) [13] explored various compression techniques for DNA data, implementing four algorithms: LZW, run-length encoding, arithmetic coding, and substitution method. Their comparative analysis revealed that while arithmetic coding achieved the best compression ratios across multiple species, run-length encoding was found unsuitable for DNA data compression. The study emphasizes the necessity of efficient storage solutions as DNA databases

continue to expand due to increasing genomic sequencing efforts.Umesh Ghoshdastider et al. [14] have introduced Genome Compress, a novel algorithm for DNA compression that effectively compresses both repetitive and non-repetitive DNA sequences. The algorithm achieves a lower bits-per-base ratio (1.25 bits/base) compared to existing methods (1.76 bits/base). The authors propose a five-bit encoding scheme that captures the repetitive nature of DNA while ensuring simplicity and speed in execution. Behshad Behzadi et al. (2005) [15] present DNAPack, a dynamic programming-based algorithm designed to address the inefficiencies of standard compression tools in handling DNA sequences. The authors emphasize the necessity for specialized algorithms that leverage the unique characteristics of DNA. DNAPack incorporates approximate repeats and complementary palindromes, demonstrating a competitive compression ratio in performance comparisons with existing algorithms.

III. PROPOSED SYSTEM

The system proposed in Figure 2 is designed to perform efficient compression of DNA sequences using two different algorithms: Huffman Encoding and Arithmetic Coding. The overall workflow begins by accepting a DNA sequence as input from the user, consisting of nucleotides (A, C, G, T).
In the first phase, the system calculates the frequency of each nucleotide in the sequence. These frequencies are then used to build a Huffman Tree by repeatedly combining the two nucleotides with the lowest frequencies into a new node, ultimately resulting in a binary tree where each nucleotide is assigned a unique Huffman Code. The DNA sequence is then encoded using these Huffman Codes, significantly reducing the memory required for storage. The system also calculates and displays the memory usage and compression ratio for Huffman Encoding by comparing the original sequence with the compressed one.

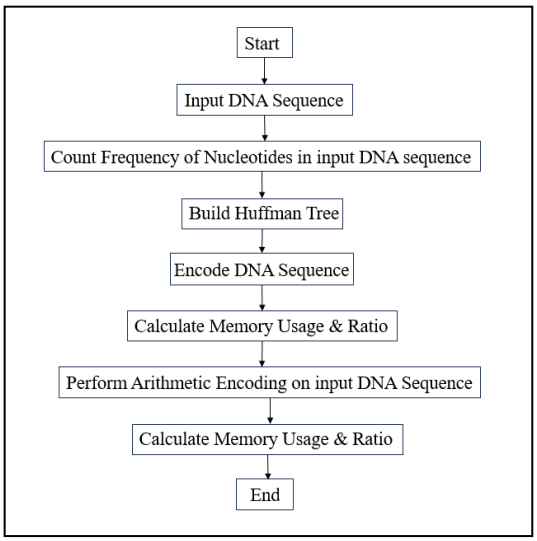

Figure 2 Workflow of System

In the second phase, the system performs Arithmetic Coding by calculating the probabilities of each nucleotide and assigning unique intervals to them based on their frequencies. The DNA sequence is encoded into a single floating-point number that lies within the combined interval of all nucleotides in the sequence. The system calculates and displays the memory usage and compression ratio for Arithmetic Coding as well, providing a comparison with Huffman Encoding.
This modular approach allows the system to seamlessly apply both Huffman and Arithmetic coding methods to compress DNA sequences, optimizing memory usage and enabling efficient storage and transmission of biological data. The system's performance is measured by calculating the time taken for both encoding methods, making it suitable for real-time applications in genomic data analysis, storage, and transmission.

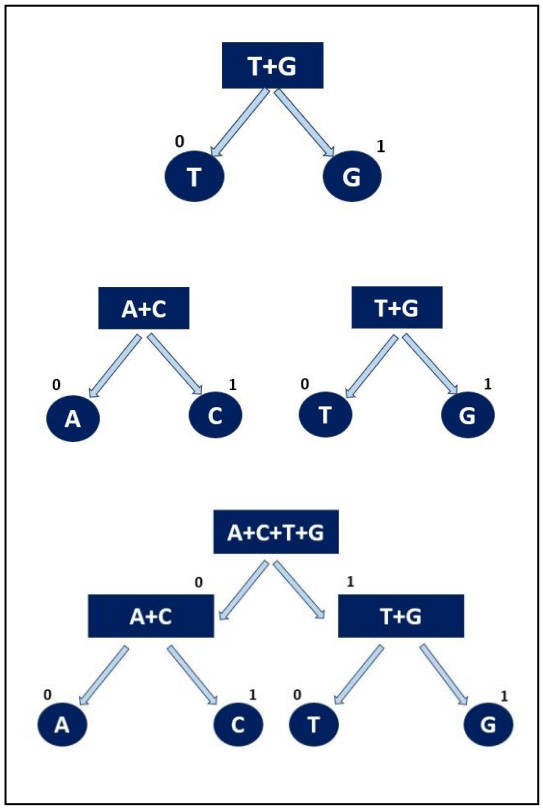

Figure 1 Construction of Huffman Tree

The following algorithm outlines the process of DNA sequence compression using both Huffman and Arithmetic coding techniques. The algorithm first builds a Huffman tree to generate binary codes for each nucleotide, followed by encoding the sequence and calculating memory usage. Additionally, it performs arithmetic encoding by calculating nucleotide probabilities and intervals for more efficient compression.

A. Algorithm

Initialization:

- Create a node structure for each nucleotide, containing the frequency and left/right child pointers.
- Create the HuffmanBST class to include the core data structures such as frequency map, Huffman

codes, node list, probability map, and intervals of arithmetic coding.

Input DNA Sequence:

- Prompt the user to enter a DNA sequence.

Step 1: Frequency Count:

- Iterate through the DNA sequence to count the occurrences of each nucleotide, 'A', 'C', 'G', 'T'.
- Store these frequencies in a frequency map (freq_map).

Step 2: Build Huffman Tree:

- Initialize nodes for 'A', 'C', 'G', and 'T' with their respective frequencies.
- Repeatedly find the two lowest-frequency nodes that have not been combined yet.
- Combine these two nodes into a new parent node, assigning the combined frequency and updating flags for left and right children.
- Add this new node back to the list of nodes.
- Repeat until there is only one node, which will be the root of the Huffman tree.

Step 3: Generate Huffman Codes:

- Traverse the Huffman tree to generate binary Huffman codes for each nucleotide ('A', 'C', 'G', 'T').
- Store these codes in the huffmanCodes map.

Step 4: Compression of DNA Sequence:

- Encode the DNA sequence using the Huffman code s into a compressed binary string.

Step 5: Memory Calculation (Huffman):

- Calculate the memory required by the original DNA sequence (8 bits per character).
- Calculate the memory required by the Huffman encoded DNA sequence (number of bits in the encoded string).
- Display the memory usage and the compression ratio.

Step 6: Huffman Coding Time:

- Measure the time taken to complete the Huffman coding process.

Step 7: Arithmetic Coding - Probability Calculation:

- Calculate the probability of occurrence of each nucleotide based on its frequency.
- Calculate cumulative intervals for each nucleotide to be used in arithmetic coding.

Step 8: Arithmetic Encoding:

- Set range [low, high] [low, high] [low, high] to be [0.0,1.0] [0.0, 1.0] [0.0,1.0].
- For each nucleotide that occurs in the sequence calculate the new range based on its corresponding interval.
- The last encoded value is the middle value of low and high.

Step 9: Memory Calculation (Arithmetic):

- Compute memory needed by the original DNA sequence (8 bits per character).
- Compute memory needed by the arithmetic encoded value (size of a double in bits).
- Display the memory usage and the compression ratio for arithmetic coding.

Step 10: Arithmetic Coding Time:

- Measure the time taken to complete the arithmetic coding process.

End:

- The program prints out Huffman codes, the coded DNA sequence, memory used, the Huffman and arithmetic coding compression ratios, and the time taken for all processes.

B. Pseudocode

```
BEGIN
   INPUT DNA sequence
   COUNT frequency of each nucleotide (A, C, G, T)

   // Huffman Encoding Process
   BUILD Huffman Tree using frequencies
   GENERATE Huffman codes for each nucleotide
   ENCODE DNA sequence with Huffman codes
   CALCULATE memory usage and compression ratio for Huffman

   // Arithmetic Encoding Process
   CALCULATE probabilities for each nucleotide
   ASSIGN cumulative intervals to each nucleotide
   PERFORM Arithmetic encoding on DNA sequence
   CALCULATE memory usage and compression ratio for Arithmetic
    OUTPUT Huffman encoded sequence, memory usage, compression ratio
   OUTPUT Arithmetic encoded value, memory usage, compression ratio
   OUTPUT time taken for both encoding methods
END
```

Figure 3 Pseudocode

IV. Memory Analysis And Compression Efficiency

The performance of both Huffman and Arithmetic coding implementations was evaluated using randomly generated DNA sequences. The Huffman-based Binary Search Tree (BST) approach assigned optimal prefix codes based on nucleotide frequency, while Arithmetic coding utilized probability intervals for encoding. We compared both methods in terms of memory efficiency and compression ratio. Huffman coding achieved significant memory savings, compressing sequences effectively with minimal computation time. Arithmetic coding required slightly more processing time while offering competitive compression ratios.

A. Memory Usage

To assess the compression efficiency, it's important to compare the memory used by the original DNA sequence and the compressed version. This comparison gives insight into how much space is saved by applying the encoding techniques and highlights the algorithm's effectiveness in reducing storage requirements.

Input Memory: Each character in the DNA sequence (8 bits for ASCII) contributes to the total memory requirement.

$$Memory\ Input = Length\ of\ DNA\ sequence\ \times 8\ bits$$

Equation 1

Output Memory: The length of the encoded string determines the memory used by the Huffman encoded sequence.

$$Memory\ Output = Length\ of\ encoded\ DNA\ bits$$

Equation 2

B. *Compression Efficiency*

Compression efficiency refers to the effectiveness of a data compression algorithm in reducing the size of the original data.
Compression Ratio: It is defined as the ratio between the compressed size of the data and the original size.

$$Compression\ Ratio = \frac{Compressed\ Size}{Original\ Size}$$

Equation 3

The following Example 1 and Example 2 illustrates the memory requirements and compression efficiency of DNA sequences using both Huffman and Arithmetic encoding methods. It demonstrates the original memory usage, the compressed output, and the resulting compression ratios for each encoding technique.

Example 1.
DNA Sequence: ATTTTGAGGAGACCCAAGTAGACA
Total number of nucleotides = 24

$$Memory\ required = 24\ nucleotides \times 8\ bits$$
$$= 192\ bits$$

Huffman Encoding:
Huffman Encoded DNA compression:
110101010110111010111011000000111110011110110011

Memory required to store compressed sequence: 48 bits

$$Compression\ ratio = \frac{48\ bits}{192\ bits} = 0.25$$

Arithmetic Encoding:
Arithmetic Encoded Value: 0.374722
Memory required to store encoded value: 64 bits

$$Compression\ ratio = \frac{64\ bits}{192\ bits} = 0.33$$

Example 2.
DNA Sequence: ATGGTGGCCTACTGGAGACAGGCTGGACTCAGCTA CATCCGATACTCCCAGATCTGTGCAAAAGTAGTGA GAGATGCACTGAAGACAGAATTCAAAGCAAATGC CAAGAAGACTTCTGGCAGCAGCGTAAAAATTGTGA AAGTAAAGAAGGAATAA

Total number of nucleotides = 156

$$Memory\ required = 156\ nucleotides \times 8\ bits$$
$$= 1248\ bits$$

Huffman Encoding:
Huffman Encoded DNA compression:
11011010011010000000111000110101110110011101000011 01011000100111000011100110100001011011100010000000 11101101000110011000111111111001111001101110111011 01100011000110111110110011101111010100111111100011 11110110000011111011111011000101000110100011100011 10001001111111111101011001101111111001111111110111 110101111011111

Memory required to store compressed sequence: 312 bits

$$Compression\ ratio = \frac{312\ bits}{1248\ bits} = 0.25$$

Arithmetic Encoding:
Arithmetic Encoded Value: 0.355425
Memory required to store encoded value: 64 bits

$$Compression\ ratio = \frac{64\ bits}{1248\ bits} = 0.0512$$

From Example 1 and Example 2, we deduce that the Huffman Algorithm can be used when the DNA sequence is compact or smaller. Huffman encoding is used wherever its output is lesser than 64 bits. In case the DNA sequence is large or contains many unique symbols, then Arithmetic encoding would favour Huffman Encoding. The arithmetic method often provides better compression ratios and hence proves to be more efficient for complex or extended datasets, though it is more complex to implement. Figure 4(a) shows the memory required for original DNA data versus after compression, with Arithmetic Coding using the least memory. Figure 4(b) presents compression ratios, showing the extent of compression, each method provides, with Arithmetic Coding achieving the highest compression ratio in Example 2. Figure 4(c) highlights memory savings for both methods, with Arithmetic Coding achieving higher bit savings across examples.

Example	Total Nucleotides	Memory Required (bits)	Memory Required for Huffman (bits)	Huffman Compression Ratio	Arithmetic Encoded Value	Memory Required for Arithmetic (bits)	Arithmetic Compression Ratio
1	24	192	48	0.25	0.374722	64	0.33
2	156	1248	312	0.250	0.355425	64	0.051

Table 1 Comparison of Memory Requirements and Compression Ratios for Huffman and Arithmetic Encoding of DNA Sequences

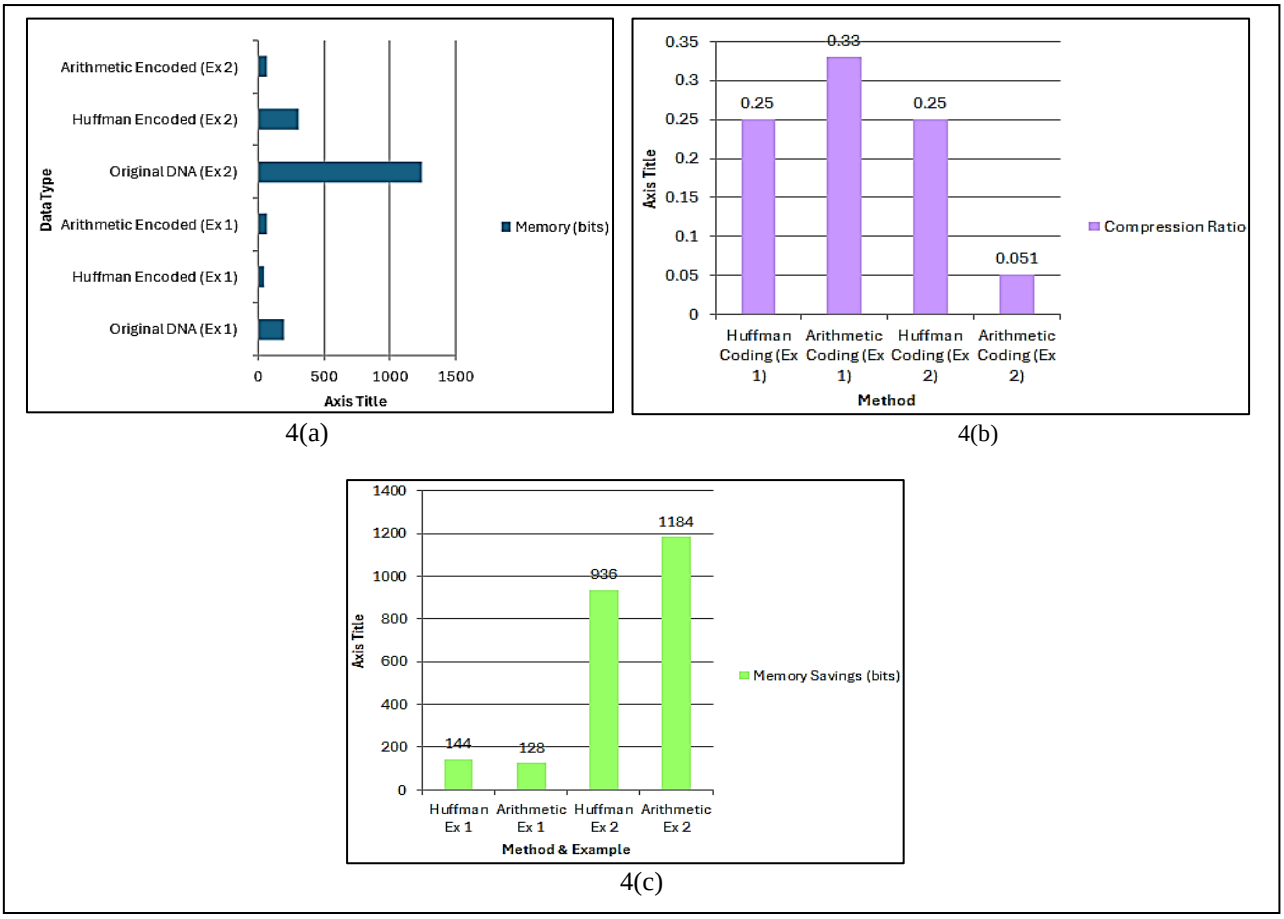

Figure 4(a): Memory Usage, 4(b): Compression Ratio 4(c): Memory Saving. Graphical Representation of efficiency of Huffman vs. Arithmetic Encoding

V. Performance And Accuracy Evaluation

To ensure the reliability of the compression methods, we assessed the accuracy DNA sequences after decompression. As shown in Figure 5 and Figure 6, both the Huffman and Arithmetic coding algorithms maintained lossless compression, meaning that the original sequences were perfectly reconstructed without any loss of information. We conducted a series of tests to compare the decompressed sequences against their original counterparts, confirming that all nucleotide sequences matched exactly. This fidelity is crucial for bioinformatics applications, where data integrity is paramount for accurate genomic analysis and interpretation.

```
Entered the DNA Sequence: AGTTCAAC
Huffman Encoded DNA Sequence: 1100101001111101

Decompression:

Huffman Decoding Steps:
Decoded 11 to A
Decoded 00 to G
Decoded 10 to T
Decoded 10 to T
Decoded 01 to C
Decoded 11 to A
Decoded 11 to A
Decoded 01 to C

Huffman Decoded DNA Sequence: AGTTCAAC
```

Figure 5 Decoding Huffman DNA Sequence

```
Entered the DNA Sequence: AGTTCAAC

Decompression:

Arithmetic Decoding Steps:
Decoded value 0.279470 to A with interval [0.000000, 0.375000]
Decoded value 0.745255 to G with interval [0.625000, 0.750000]
Decoded value 0.962036 to T with interval [0.750000, 1.000000]
Decoded value 0.848145 to T with interval [0.750000, 1.000000]
Decoded value 0.392578 to C with interval [0.375000, 0.625000]
Decoded value 0.070312 to A with interval [0.000000, 0.375000]
Decoded value 0.187500 to A with interval [0.000000, 0.375000]
Decoded value 0.500000 to C with interval [0.375000, 0.625000]

Arithmetic Decoded DNA Sequence: AGTTCAAC
```

Figure 6 Decoding Arithmetic DNA Sequence

VI. Application And Use Cases

A. *Huffman Coding*

- Application

1. Data Compression: Huffman coding is widely used in lossless data compression algorithms. It's employed in formats like ZIP, and GZIP, and compression standards such as JPEG for images.
2. Multimedia Files: JPEG and MP3 use Huffman coding for image and audio data compression.
3. Text Compression: When encoding text files, frequently occurring characters are given shorter codes, while infrequent ones get longer codes.

- Use Cases

1. File Compression: In file compression tools like ZIP or GZIP, Huffman coding is part of the algorithm that compresses files.
2. Image Compression: In JPEG, after transforming the image using DCT (Discrete Cosine Transform), Huffman coding is applied to further compress the transformed data.
3. Communication Systems: In error detection and correction in digital communication, Huffman coding is used to reduce redundancy.

B. Arithmetic Coding

- Application

    1. Data Compression: Arithmetic coding is also used in lossless data compression but is typically more efficient than Huffman coding because it can encode symbols into fractional bit lengths.
    2. Video and Audio Compression: Modern multimedia codecs like H.264 (used in video streaming, Blu-ray discs, etc.) and HEVC (High-Efficiency Video Coding) make use of arithmetic coding to efficiently compress data.
    3. Error Resilient Systems: Arithmetic coding is sometimes preferred in systems that need to correct small errors in compressed data, as it can compress data closer to the entropy limit.

- Use Cases

    1. High-efficiency Compression: Arithmetic coding can provide slightly better compression ratios than Huffman, especially in situations where symbol probabilities differ only slightly.
    2. Multimedia Codecs: Used in advanced codecs like H.264, H.265 (HEVC), and AV1 for efficient video compression, allowing for high-quality streaming at lower bit rates.

    3. File Compression Formats: Some advanced file compression systems may use arithmetic coding for better performance than Huffman in certain situations, such as in the JBIG2 image compression standard.

VII. CONCLUSION

In this paper, we presented a comparative analysis of Huffman and Arithmetic Coding for the compression of DNA sequences, addressing the growing need for efficient data management in the field of bioinformatics. Our implementations demonstrated that both techniques effectively reduce the memory footprint of genomic data while maintaining lossless accuracy. The Binary Search Tree (BST)-based Huffman coding provided faster encoding times, while Arithmetic coding achieved superior compression ratios for datasets with non-uniform nucleotide distributions. The results underscore the potential of these compression methods to enhance genomic data storage and [1]transmission capabilities, making them valuable tools for researchers working with large-scale genomic datasets. Future work may include using hybrid approaches that leverage the strengths of both techniques further and enhance results in various bioinformatics tasks. Future directions include optimizing Huffman and Arithmetic Coding for real-time applications like streaming and IoT, combining them with AI for versatile compression, enhancing error resilience for unreliable networks, improving compression for high-resolution media formats (e.g., 8K, VR), and exploring their potential in quantum and advanced computing.

ACKNOWLEDGEMENT

The authors wish to extend their gratitude to Mrs. Minal Deshmukh for their guidance and support, throughout this research project. Their expertise and encouragement were instrumental in the completion of this study. We also want to acknowledge Vishwakarma Institute of Information Technology for providing the resources and infrastructure, for our research efforts.

VIII. REFERENCES

[1] Gurunathan, A., & Moideen, F. B. K. (2024). Enhanced Index Based DNA Sequence Compression Algorithm. *International Journal of Intelligent Engineering & Systems*, *17*(1).

[2] Ali Hassan ,Sadaf Javed ,Sajjad Hussain, Rizwan Ahmad & Shams Qazi (2024).Arithmetic N-gram: an efcient data compression technique.

[3] Babita Kumari, Neeraj Kumar Kamal, Arif Mohammad Sattar & Mritunjay Kr. Ranjan(2023). Adaptive Huffman Algorithm for Data Compression Using Text Clustering and Multiple Character Modification.

[4] Hussein, A. A., Al Baity, R. M., & Hadi, S. A. (2023). Randomized Information Hiding in RGB Images Using Genetic Algorithm and Huffman Coding. *Revue d'Intelligence Artificielle*, *37*(6), 1435-1440.

[5] Alosta, M., & Souri, A. (2022). Design of Effective Lossless Data Compression Technique for Multiple Genomic DNA Sequences

[6] Mehedi Hasan Sarkar and Adnan Ferdous Ashrafi(2022). Genetic Sequence Compression Using Machine Learning and Arithmetic Encoding Decoding Technique

[7] Gomathi, D., Rajathi, I., & Gopala Krishnan, C. (2021). DNA Compression Using RLE and Huffman Encoder

[8] Ganapathy, M. (2021). IT Reduction-Based Compression Algorithm for DNA Sequences.

[9] Zhao, J., Ma, Z., & Xie, F. (2021). An Improved Huffman-Based Compression Algorithm for Large-Scale DNA Sequences

[10] Mishra, P., Bhaya, C., Pal, A. K., & Singh, A. K. (2021). Compressed DNA Coding Using Minimum Variance Huffman Tree

[11] Pulova-Mihaylova, B., Mihaylov, I., Avdjieva, I., & Vassilev, D. (2024). A System for Compression of Sequencing Data. International Journal of Intelligent Engineering & Systems, 17(1)

[12] Alam Jahaan, Dr. T.N. Ravi, Dr. S. Panneer Arokiaraj (2017). Comparative Study and Survey on Existing DNA Compression Techniques.

[13] Machhi, V., & Patel, M. S. (2016). Compression Techniques Applied to DNA Data of Various Species. International Journal of Bio-Science and Bio-Technology, 8(3), 45-52.

[14] *Ghoshdastider, U., & Saha, B. (2005). GenomeCompress: a novel algorithm for DNA compression.Behshad Behzadi, Fabrice Le Fessant (2005).*DNA Compression Challenge Revisited: A Dynamic Programming Approach

[15] Behzadi, B., & Le Fessant, F. (2005, June). DNA compression challenge revisited: a dynamic programming approach. In *Annual Symposium on Combinatorial Pattern Matching* (pp. 190-200). Berlin, Heidelberg: Springer Berlin Heidelberg.