Genome Compression Using Program Synthesis

Denys Maletskyi[a] and Volodymyr Shymanskyi[a]

[a] *Lviv Polytechnic National University, Lviv, Ukraine*

Abstract
With the growing amount of genomic data, finding efficient compression methods is crucial for both storing and analyzing this data. This article discusses the importance of genome data compression, especially in large genomic projects. It introduces program synthesis as a powerful tool for data compression. Program synthesis can create programs that efficiently represent and reproduce data, making it a promising approach for compressing genomes. Readers will learn about program synthesis, how it works, and how it can be used for genome data compression. The article shows how we can use program synthesis to compress genome sequences better. Although the focus is on genomic data, the methods and insights shared in this article can also be applied to compressing other types of data, offering a novel perspective to the current discourse on data compression strategies.

Keywords 1
Program synthesis, compression, encoding, decoding, genome, DNA

1. Introduction

With the rapid progression of high-throughput sequencing technologies, the amount of genomic data available has surged exponentially, necessitating the development of efficient genome compression methods. As the deluge of genomic data continues, the importance of devising innovative and effective genome compression techniques cannot be overstated.

Recent literature unfolds various approaches to genome compression. Reference-based compression algorithms, such as CRAM [1] and RENANO [2], present a noteworthy approach to genome data compression. These methodologies achieve substantial compression rates by utilizing a reference genome to represent individual sequences efficiently. While promising in their compression efficacy, these algorithms carry the significant drawbacks of extended compression time and considerable memory requirements, aspects that are detrimental in scenarios demanding swift data processing and limited computational resources. These limitations highlight the necessity for novel strategies to balance compression efficiency, speed, and resource consumption to facilitate more feasible and practical applications in large-scale genome sequencing projects.

Alternative methods, such as CMIC [3], opt for a different balance, slightly relinquishing the compression ratio to gain random access functionality, a feature indispensable for a wide array of tasks. This type of approach is intriguing; it provides a distinct perspective on tackling genome compression challenges, offering a mix of functionality and efficiency. In light of this, our work introduces an alternative approach, discussed in section 4.2.2, that guarantees a fixed compression ratio of 3x. This proposed method not only preserves the vital characteristic of random access but also stands out for its simplicity and unprecedented speed in the compression process, making it an attractive option for tasks that require rapid and efficient handling of genomic data.

Another category of approaches, such as the one elucidated in "Genome Compression Against a Reference" [4], concentrates on batch compression of genomes. These methodologies leverage the inherent similarities among human genomes, as the genetic variation between individuals is relatively minimal. Hence, these techniques primarily focus on compressing only the sections of the genome data

IDDM'2023: 6th International Conference on Informatics & Data-Driven Medicine, November 17 - 19, 2023, Bratislava, Slovakia
EMAIL: denys.maletskyi.mknssh.2022@lpnu.ua (A. 1);vshymanskiy@gmail.com (A. 2)
ORCID: 0009-0005-4167-4858 (A. 1); 0000-0002-7100-3263 (A. 2)

that exhibit differences. By doing so, these approaches efficiently handle the voluminous data associated with multiple genomes, ensuring that the redundancy in the information is effectively minimized, leading to efficient storage and management of genomic datasets.

A distinct method is presented in "Cryptography and Reference Sequence Based DNA/RNA Sequence Compression Algorithms" [5], where the authors ingeniously apply principles from one domain to another. In this novel approach, techniques traditionally used in cryptography are employed for the purpose of genome compression. This cross-domain application opens up new possibilities, offering a unique perspective and methodology in addressing the challenges of genome data compression. Through the integration of cryptographic principles, the proposed approach ensures efficient compression. It introduces innovative mechanisms for securing and managing genomic data, thereby broadening the horizon of genome data compression strategies.

The surveyed approaches underscore the swift advancement unfolding within the genome compression research sector. Observing this rapid progression, it is pivotal to introduce innovative strategies that further the field's capabilities. In response to this imperative, we put forward a novel approach in our study that incorporates a technique hitherto unexplored in the realm of genome compression. This innovative method centers on program synthesis, particularly emphasizing equality saturation for compression tasks. The employment of program synthesis, and more specifically, the equality saturation approach, is posited to offer commendable performance in genome compression endeavors, opening new avenues for efficient and effective genomic data handling and analysis.

2. Background

Data compression is pivotal for minimizing data size, facilitating efficient storage, and expediting transmission. It's essential for reducing costs and quickening the pace of data analysis and sharing. Data compression methodologies fall into two categories: lossless and lossy. Lossless compression ensures that the original data's integrity is maintained upon decompression, whereas lossy compression results in some data loss but often attains higher compression rates 1.

Shannon's entropy [6] is integral to data compression, providing a theoretical limit to the best possible lossless compression ratio achievable by any compression algorithm. Grasping the concept of Shannon's entropy is vital as it provides insights into data's compressibility, steering the creation of more efficient algorithms. Shannon's entropy is calculated using the formula:

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

Traditional compression algorithms, including Huffman encoding [7], Arithmetic encoding [8], Run-Length Encoding (RLE) [9], and dictionary-based methods like LZW algorithm [10] or GZIP [11], serve as foundational knowledge for this discussion. Huffman encoding generates variable-length codes for each symbol based on their frequencies, while Arithmetic encoding represents a sequence of symbols as a single number. Both RLE and ZIP exploit redundancy in data for compression.

2.1. Genome compression challenges

Genome data compression is daunting due to the genomic dataset's unique and voluminous nature. Here are some of these challenges:

- Volume: A single human genome encompasses billions of base pairs, resulting in substantial data volumes, necessitating effective compression algorithms5.
- Diversity: The inherent diversity in genomic data means that variations between genomes are vast, necessitating algorithms that can efficiently handle a broad data spectrum without losing crucial information6.
- Noise: Sequencing errors and N's (unknown nucleotides) in the data introduce noise, complicating the compression process7.
- Accessibility: The compressed data should be easily retrievable for further analysis and research without imposing significant computational overheads8.

Addressing these challenges is crucial for devising effective genome data compression algorithms.

2.2. Program synthesis overview

Program synthesis is the process of automatically generating programs based on a given specification. Instead of manually crafting algorithms, developers can provide requirements, and a synthesizer outputs a program matching these criteria [7]. Program synthesis, in essence, facilitates automating parts of the programming process.

There are multiple approaches to program synthesis:

1. Deductive Synthesis: Starts with a high-level specification and refines it step-by-step into a full-fledged program [12].
2. Stochastic Synthesis: Uses probabilistic methods to explore the program space, often making use of genetic algorithms [13].
3. Syntax-Guided Synthesis (SyGuS): Defines the semantic constraint the desired function must satisfy and a syntactic constraint restricting the space of allowed implementations [14].

A fundamental concept tied with program synthesis is Kolmogorov complexity [15], which quantifies the computational resources needed to specify a string or piece of data. For a string s, its Kolmogorov complexity $K(s)$ is the length of the shortest possible program (in some fixed programming language) that, when run, produces s as output [16].

$$K(s) = min_{p \in U(p)} = |p| \tag{2}$$

Where U is a universal Turing machine, and p is a program such that $U(p) = s$.

Equality saturation is an approach wherein a given program's many equivalent forms are explored simultaneously. It employs e-graphs to represent a large set of programs compactly. E-graphs allow for finding rewrites in parallel, enabling the synthesizer to search for the most optimal (shortest or fastest) program that satisfies the same specification as the original [17].

3. Methodology

The Methodology chapter outlines the research design, detailing the specific methods and procedures for data collection, analysis, and experimentation. This chapter serves as a blueprint for the entire research process, providing readers with a comprehensive understanding of the approach to addressing the research questions.

3.1. Analysis of Entropy in Human Genome Data

The data under consideration in this research is the human genome, primarily composed of four nucleotides: Adenine (A), Cytosine (C), Guanine (G), and Thymine (T). However, it's crucial to note that biological data acquisition, particularly the digitalization of DNA genome sequences, is not entirely error-free. As a result, certain parts of the genome are marked as 'N' to indicate uncertainties or errors in sequencing. These 'N' markers introduce additional complexity when analyzing the genomic data.

The entropy of the human genome is evaluated through the following steps:

1. Average Frequency Calculation: For each nucleotide s ∈ {A, C, G, T, N}, the average frequency fs is calculated based on its occurrences in the entire dataset.
2. Average Probability Calculation: The average probability $P(s)$ of each nucleotide is then determined by normalizing its frequency fs by the total number of nucleotides in the dataset.
3. Entropy Calculation: Using Shannon's entropy formula (Shannon, C.E., 1948), the entropy H is then calculated as

$$H = \sum_{s \in \{A,C,T,G,N\}} P(s) log_2 P(s) \tag{3}$$

Based on the analysis conducted, the average probabilities for each of the nucleotides in the dataset demonstrated in Table 1:

Table 1
Calculated probabilities of each symbol in the human genome based on their frequencies in a dataset.

Nucleotide	A	C	G	T	N
P(X)	0.279	0.194	0.195	0.280	0.048

Using these probabilities, Shannon's entropy H can be calculated using the formula:

$$H = -(P(A)log_2P(A) + P(C)log_2P(C) + P(G)log_2P(G) + P(T)log_2P(T) + P(N)log_2P(N)) \quad (4)$$

Substituting the given probabilities:

$$H = -(P(0.279)log_2P(0.279) + P(0.194)log_2P(0.194) + P(0.195)log_2P(0.195) + P(0.280)log_2P(0.280) + P(0.048)log_2P(0.048)) \quad (5)$$

$$H \approx 2.162\, bits \quad (6)$$

Therefore, the calculated entropy of the dataset based on the given probabilities is approximately 2.162 bits. This value serves as a theoretical lower limit for the average number of bits required to represent each nucleotide in the human genome for the given dataset. It provides a benchmark for evaluating the effectiveness of different compression algorithms applied to this specific genomic data.

By understanding the entropy inherent in the human genome, particularly with the considerations of data errors denoted by 'N', we can better gauge the challenges and limits of compressing this form of biological data. It sets the stage for exploring advanced compression algorithms tailored for genomic data.

3.2. Reducing Entropy Through Data Transformation

3.2.1. Entropy Calculation for Combination-based String

One of the pivotal ideas in data compression is that entropy can change based on how data is represented. For instance, consider a simple transformation where consecutive identical symbols are replaced with an integer representing the length of the sequence which is also important with low technical characteristics of the system [18] and optimization algorithms using parallel computing [19]. The entropy H′ of this new data would typically be greater than the entropy H of the original data due to the increased number of unique symbols (combinations) in the transformed string. However, because these new unique symbols represent 2-symbol combinations from the original string, the effective bits per original symbol would be $\frac{H\prime}{2}$, which could be more efficient for compression.

3.2.2. Focusing on Programmatic Representations

This article aims to shift the spotlight from conventional transformations to an alternate paradigm: representing data as a program. Here, the concept of Kolmogorov complexity comes into play, which posits that the complexity of a data string can be defined as the length of the shortest possible program that can generate that string. Thus, if we can find a shorter programmatic representation, we can effectively lower the complexity and, thereby, the entropy of the original data.

To illustrate this, let's consider the string "AAAAAAAACCCCCCCC". The entropy of this data string, considering its highly repetitive nature, would be:

$$H = -\big(P(A)log_2P(A) + P(C)log_2P(C)\big) = 1\, bit \quad (7)$$

Compression algorithms that focus solely on entropy, like Huffman encoding, would be highly effective here, compressing the data to just 16 bits: "0000000011111111".

Now, consider a Python representation of the same string:

'A' + 'A' + 'A' + 'A' + 'A' + 'A' + 'A' + 'A' +
'C' + 'C' + 'C' + 'C' + 'C' + 'C' + 'C' + 'C' (8)

By applying mathematical axioms, we can rewrite this program into a shorter one:

'A' * 8 + 'C' * 8 (9)

Going further, this can be rewritten as:

('A' + 'C') * 8 (10)

This can then be further reduced to its Reverse Polish Notation (RPN):

$$AC+8* \tag{11}$$

When we apply Huffman encoding to this RPN representation, we get a string with an average code length L of $\frac{12}{16} = 0.75\ bits$, which is indeed less than the original entropy H = 1 bit of the original string.

This example vividly demonstrates that by shifting our focus to different data representations—in this case, programmatic representations—we can achieve even more efficient compression than what would be suggested by the original data's entropy. This opens up a new frontier for data compression techniques, especially in specialized fields like genomic data compression.

3.3. Program Synthesis for Data Transformation

3.3.1. Introduction to the Problem Space

We have established that altering data representations can reduce entropy, enhancing the compression ratio. Program synthesis, particularly focusing on the method of equality saturation, offers a powerful means to discover optimal programs capable of regenerating the original data. Tools like EGG (E-Graphs Good) [20] provide robust data structures designed for efficient exploration of possible rewrites for such programs.

However, the naive approach of converting an entire genomic sequence into a complex mathematical construct—as illustrated in the previous section—is computationally untenable. This method would require enormous processing power and time, rendering it practically useless for large-scale genomic data.

3.3.2. The Challenges of Sequence Length

One could consider a more feasible approach, where all possible combinations of a fixed length n nucleotides are identified, and an optimal program is sought for each combination. Yet this approach presents its own set of challenges, the most formidable of which is computational complexity.

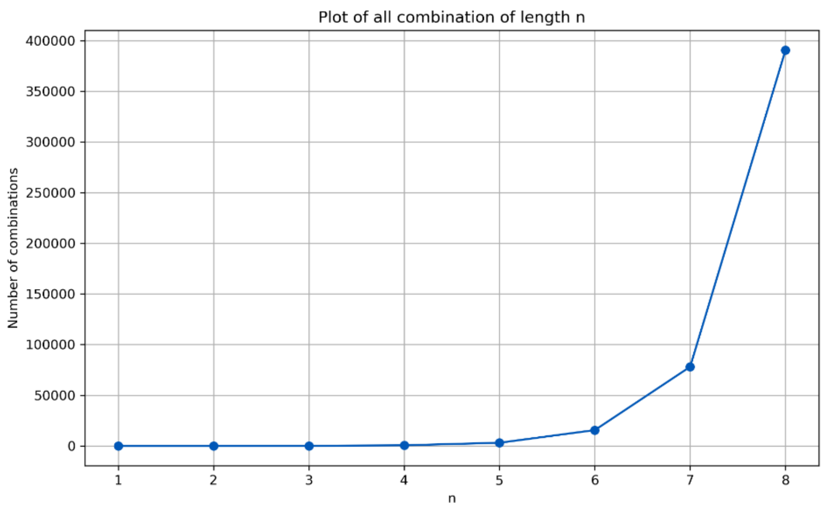

Figure 1: Visualization of exponential growth of all possible combinations of nucleotides of length n.

The total number of such combinations would be 5n, where n is the length of the nucleotide sequence under consideration (Fig. 1). Because the number of combinations increases exponentially with n, we would quickly hit a computational ceiling. This is a significant issue because longer sequences generally offer greater opportunities for effective compression, making it imperative to find a strategy that allows us to consider long sequences without prohibitive computational costs.

3.3.3. Toward a More Scalable Approach

Given these challenges, it becomes evident that a different strategy must be devised. One that enables us to explore longer nucleotide sequences without overwhelming computational resources while also effectively lowering the entropy of the genomic data to achieve a high compression ratio.

To develop a more efficient method, we can capitalize on the similarities between different nucleotide combinations in the context of program synthesis. Consider sequences like "AAAA" and "CCCC"; their programmatic representations would be quite similar, e.g., ‘A’ × 4 and ‘C’ × 4. These similarities hint that we might not need to consider every possible combination in the search space, offering a potential route to computational savings.

3.3.4. Run-Length Encoding as a Metric

One effective metric for gauging the similarity between sequences is Run-Length Encoding (RLE) (Tsukiyama, 1996). When we apply RLE to a sequence, we reduce it to a series of counts representing each distinct element's lengths of runs. For example, the sequence "AAAA" would be transformed into the count (4), and "CCCC" would also yield (4).

By analyzing the sequences using RLE, we begin to notice that many combinations share identical or similar counts. These counts effectively serve as "fingerprints" that allow us to cluster similar sequences together, significantly reducing the search space for program synthesis.

By employing a metric like RLE to identify and group similar sequences, we can substantially reduce the number of unique combinations that must be analyzed. This provides a two-fold benefit: first, it allows the computational resources to focus only on distinct "classes" of sequences, making the process more tractable. Second, it still allows us to consider longer sequences without incurring exponential computational costs, thereby offering the potential for more effective compression.

3.3.5. Tailoring the Search Space with Subset Sum

To achieve even greater computational efficiency, we can move beyond merely reducing the search space of similar sequences and directly focus on sequences most pertinent to our objective. A compelling avenue for this targeted approach is to examine the unique Run-Length Encoding (RLE) [4] counts that could occur for any combination of nucleotides of size n.

To generate these unique RLE counts, we can adapt the well-known Subset Sum problem—a combinatorial problem that aims to find a subset whose sum matches a given target number. In our case, the target number is n, the length of the nucleotide sequence.

The standard Subset Sum problem seeks to find a subset of numbers that sum up to a given target. However, for our application, we need to introduce a slight modification: in addition to finding the subset that adds up to n, we also need to consider the permutations of this subset. This is critical because different permutations of the same subset can correspond to different length-equivalent nucleotide sequences. For instance, the RLE counts (2, 1, 1) and (1, 2, 1) will correspond to different sequences, for instance, "AACG" and "CAAG".

This modified Subset Sum approach allows us to efficiently generate all unique RLE counts that could exist for sequences of length n. By synthesizing programs only for these unique counts, we can dramatically cut down on the computational resources needed, as we are now focusing solely on the 'representative' sequences that capture all possible RLE patterns for the given length.

The benefit is twofold: first, it narrows down the search space to only those sequences that are absolutely necessary for program synthesis, making the approach far more computationally efficient. Second, it does so without compromising the integrity of the analysis, as these unique RLE counts still provide a comprehensive overview of all the possible sequence patterns for a given length.

In conclusion, this targeted computation method that utilizes a modified Subset Sum problem to focus on unique RLE counts offers a highly efficient and effective way to apply program synthesis for genomic data compression. By selectively focusing on relevant sequences, we can achieve high compression rates while significantly reducing computational costs.

3.4. Implementing Program Synthesis for Compression

In the context of our genome compression task, we concentrate primarily on sequences represented through Run-Length Encoding (RLE) [9]. RLE produces tuples of values and counts, where values denote nucleotides, and counts represent their consecutive repetitions. With our focus narrowed down to these RLE sequences, the complexity of the task is significantly reduced.

3.4.1. Value Encoding

Given that RLE sequences inherently lack consecutive identical symbols, we allocate a mere two bits to encode each nucleotide value, effectively representing {'A', 'C', 'G', 'T', 'N'} \ {previos symbol}. This minimalist encoding strategy offers simplicity and efficiency, aiding the overall compression process without adding unnecessary computational overhead.

3.4.2. Count Encoding through Program Synthesis

The count component in the RLE sequences presents a more dynamic and diverse range of values, necessitating a thoughtful approach for efficient encoding. In response to this challenge, our methodology employs program synthesis—specifically, the technique of equality saturation utilizing E-Graphs.

Equality saturation through E-Graphs [13] provides a robust platform for exploring various equivalent forms of a given count, facilitating the identification of the most efficient representation. The process involves the generation of multiple program alternatives, with each variant being a potential candidate for the optimal program. Through systematic exploration and comparison, the most concise and efficient program is selected for representing a given count.

3.4.3. Program Encoding

Upon synthesizing optimal programs, the next step involves their encoding. Here, Huffman encoding serves as a primary tool renowned for its effectiveness in minimizing the length of encoded data. However, in cases where further compression is desirable, we resort to a custom packaging algorithm. This tailor-made algorithm provides an additional layer of compression, subtly improving upon the results achieved through Huffman encoding alone.

We construct a comprehensive genome compression framework through this multi-tiered approach, combining value encoding, program synthesis for count representation, and subsequent encoding of synthesized programs. This framework offers promising compression ratios and ensures efficiency and speed in both the compression and decompression stages.

4. Results and Discussion

4.1. Data Set Selection

For the purpose of this research, we utilize publicly accessible human genome sequences. These sequences have been obtained from two primary repositories: the National Center for Biotechnology Information (NCBI) [21] and the Genome Aggregation Database (gnomAD) [22]. The selection of data for this study has been influenced by specific criteria, which we outline below:

1. Completeness

It is imperative that the genome data used for this research be as complete as possible. Completeness in this context means that the sequences are well-annotated, with all regions clearly marked and identified. Additionally, the data should be free from missing values. Missing values could potentially skew the outcomes and introduce unnecessary noise into the program synthesis and compression process. Therefore, only datasets that fulfill these stringent requirements of completeness are included in the study.

2. Diversity

Given that the human genome exhibits natural variations across different ethnic and geographical backgrounds, including as much diversity as possible in our dataset is essential. By incorporating genomic sequences from various ethnic backgrounds, the study aims to account for this natural variability. This ensures that the developed compression techniques are generalizable and effective across a broad spectrum of genomic data.

In adhering to the completeness and diversity criteria, we have compiled a dataset consisting of 1000 diverse human genomes. Each genome in this dataset is meticulously selected to represent a variety of ethnic and geographical backgrounds, ensuring a comprehensive and inclusive data pool.

The dataset exhibits a balanced distribution of nucleotides. On average, each genome is approximately 3.07 GB in size. Given this, the cumulative size of our dataset is approximately 3.1 TB, providing a substantial data pool for robust testing and validation of the proposed compression techniques.

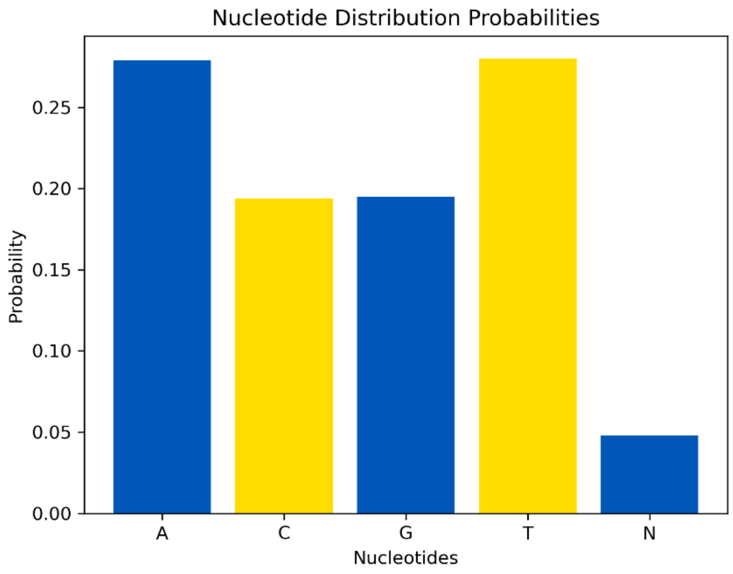

Figure 2: An analysis of nucleotide distribution within the dataset.

The probability of having 'N' is relatively low, with a value of 0.048. Given this low probability, the focus should be placed on efficiently compressing the primary nucleotides (A, C, G, T) with higher occurrence rates.

Furthermore, an analysis of the nucleotide distribution (Fig. 2) within the dataset reveals an average composition reflective of general human genome characteristics, providing a solid foundation for developing and testing our program synthesis-based compression techniques.

4.2. Benchmarks algorithms

In the initial phase of this study, it is crucial to establish a strong baseline against which the efficacy of the proposed program synthesis-based compression techniques can be measured. To that end, we have implemented and tested several classical compression algorithms to serve as benchmarks. These include Huffman and Arithmetic encoders and Run-Length Encoding (RLE).

4.2.1. Classical Algorithms

The classical algorithms, such as Huffman and Arithmetic encoders, provide a foundational understanding of the limits and potentials of general-purpose compression techniques. The RLE algorithm was also included to cover a broader spectrum of existing compression methods.

4.2.2. Custom Packing Algorithm

In addition to these well-established algorithms, we have also developed a custom packing approach specifically tailored for genomic data. The crux of the custom packing algorithm revolves around the ability to compress a combination of three nucleotides, along with the error marker 'N', into just a single byte, thereby achieving a 3-fold compression ratio and an average code length of 2.667 bits.

This approach is noteworthy for achieving a 3-fold compression ratio while maintaining the critical feature of O(1) random access. This implies that any segment of the compressed genomic data can be accessed in constant time, which is a significant advantage for many genomic applications.

The custom packing approach presents enormous potential for parallelization. Initial tests have shown compelling results in terms of both time efficiency for encoding and decoding operations. This makes it a strong candidate for comparison with the program synthesis-based techniques developed in this thesis.

To achieve this, the byte is split into distinct sections of bits, where the first two bits act as a counter for the 'N' symbol's occurrences within a nucleotide triplet. Depending on the counter's value, the remaining six bits are parsed differently:

- For N = 0:

$$\underbrace{NN}_{\textbf{Counter bits}}\ \underbrace{XXYYZZ}_{\textbf{Nucleotide bits}}$$

Here, XX, YY, and ZZ each represent one of the nucleotides in the set {A, C, G, T}, using two bits per nucleotide.

- For N = 1:

$$\underbrace{NN}_{\textbf{Counter bits}}\ \underbrace{II}_{\textbf{Index bits}}\ \underbrace{XXYY}_{\textbf{Nucleotide bits}}$$

Here, II specifies the position of 'N' in the triplet, and XX and YY encode the other two nucleotides, In this case, II represents the position of the non-'N' nucleotide, which XX encodes, and the last two with two bits for each.

- For N = 2:

$$\underbrace{NN}_{\textbf{Counter bits}}\ \underbrace{II}_{\textbf{Index bits}}\ \underbrace{XX}_{\textbf{Nucleotide bits}}\ 00$$

bits are not used.

- For N=3:

All symbols in the triplet are 'N', so the six bits following the counter are disregarded:

$$\underbrace{NN}_{\textbf{Counter bits}}\ \underbrace{000000}_{\textbf{Ignored bits}}$$

Here, XX, YY, and ZZ each represent one of the nucleotides in the set {A, C, G, T}, using two bits per nucleotide.

This structured bit-partitioning efficiently represents nucleotide triplets while considering the prevalence of 'N' symbols, streamlining the encoding and decoding processes.

Through this bit-partitioning strategy, the custom packing algorithm is not only efficient in terms of compression ratio but also maintains O(1) random access, which is pivotal for many genomic data applications. It is worth mentioning that this algorithm does not require any additional lookup table or tree for data decoding, further enhancing its performance and making it highly suited for real-time applications.

4.3. Metrics for Evaluation

Evaluating the efficiency of data compression algorithms necessitates the use of precise and reliable metrics. This research utilizes two primary metrics: Average Code Length and Compression Ratio. These metrics are crucial for quantifying the performance of the implemented compression algorithms and providing an objective comparison among them.

4.3.1. Average Code Length (L)

The Average Code Length is a vital metric that reflects the average length of the codes generated by a compression algorithm. For a set of symbols S with respective probabilities P, the Average Code Length (L) is calculated as:

$$L = \sum_{i=1}^{n} p_i \cdot l_i \tag{12}$$

Where:

- p_i : the probability of the i^{th} symbol in the dataset.
- l_i : length of the code assigned to the i^{th} symbol.
- n: total number of unique symbols in S.

The average code length provides insights into the efficiency of a compression algorithm. Ideally, the average code length should be as close as possible to the entropy of the source to achieve optimal compression.

4.3.2. Compression Ratio (R)

Compression Ratio is another crucial metric that measures a compression algorithm's reduction in data size. It is defined as the ratio of the uncompressed data size (U) to the compressed data size (C). The formula for calculating the Compression Ratio is

$$R = \frac{C}{U} \tag{13}$$

Where:

- U: Size of the uncompressed data.
- C: Size of the compressed data.

A higher compression ratio indicates more efficient data reduction. However, it is essential to consider that a higher compression ratio does not always imply better compression, as the quality and readability of the compressed data must also be maintained.

4.4. Results Comparisons

The algorithms were tested on a custom dataset of human genomes, meticulously curated and discussed in previous sections, to ensure a balanced representation of various genomic sequences. Each genome in this dataset represents a unique and diverse genetic structure, offering a challenging and comprehensive ground for testing the efficiency of the algorithms.

For each genome in the dataset, we applied the various compression algorithms: Entropy, Huffman, RLE (Run-Length Encoding), Custom Packing, and Program Synthesis combined with Huffman encoding (PS + Huffman). The Average Code Length (L) and Compression Ratio (R) were computed for each algorithm on every genome sequence, and the results were then averaged over the entire dataset to obtain a more generalized and robust understanding of each algorithm's performance.

Table 2
Comparison of benchmarks and proposed algorithm

Approach	Average Code Length (L)	Compression Ratio (R)
Entropy	2.039	3.923
Huffman	2.241	3.569
RLE	5.633	1.428
Custom Packing	2.667	3.000
PS + Huffman	**2.154**	**3.714**

The proposed approach (PS + Huffman) emerges as the most efficient method among those analyzed, closely approximating the theoretical entropy and achieving the highest compression ratio.

Its success can be attributed to the effective combination of program synthesis for count representation and Huffman encoding, which collectively optimize the compression process.

In comparison which is demonstrated in Table 2, while the Huffman algorithm and custom packing method exhibit commendable performances, they do not reach the efficiency level of the proposed approach. Although RLE is a straightforward method, it significantly lags in terms of compression ratio and code length, deeming it unsuitable for optimal genome data compression in this context.

5. Conclusions

This article has presented promising results for genome data compression through the innovative use of program synthesis, specifically employing the equality saturation technique. Our approach offers substantial improvements in compression ratio and ensures a reduction in average compression time, proving its effectiveness and efficiency in handling genome data.

The method introduced applies program synthesis meticulously to extract and efficiently encode optimal programs for different sections of genome data. This careful application bridges the gap between theoretical potential and practical utility, resulting in a robust and efficient compression technique.

Future work can enhance this approach by incorporating deeper insights into the intrinsic characteristics of genome data. The encoding phase outlined in this study also offers scope for refinement. By adopting advanced encoding strategies and integrating domain-specific knowledge, we anticipate the development of a more powerful and refined technique for genome data compression.

In conclusion, this paper marks a significant step forward in the dynamic field of genome data compression, opening up new possibilities for further research and innovation in developing more efficient compression techniques.

6. Acknowledgments

I would like to express my deep gratitude to Lviv Polytechnic National University, Lviv, Ukraine, for their unwavering support and provision of essential resources necessary for conducting this research. The university has provided a conducive and stimulating environment, which has been pivotal in the successful completion of this study.

In addition, a sincere acknowledgment is due to the Ukrainian Armed Forces. The opportunity to undertake and complete this research amidst these challenging times is largely attributed to their resilient spirit and unfaltering commitment to peace and stability in Ukraine.

7. References

[1] A. Al Ali, et al. “CRAM compression: practical across-technologies considerations for large-scale sequencing projects,” bioRxiv, pp. 2022–12, 2022.

[2] G. Dufort Álvarez, et al. “RENANO: a REference-based compressor for NANOpore FASTQ files,” Bioinformatics, vol. 37, no. 24, pp. 4862–4864, 2021.

[3] H. Chen, et al. “CMIC: an efficient quality score compressor with random access functionality,” BMC Bioinformatics, vol. 23, no. 1, p. 294, Dec. 2022, doi: 10.1186/s12859-022-04837-1.

[4] A. Laud, et al. “Genome Compression Against a Reference,” arXiv preprint arXiv:2010.02286, 2020.

[5] S. P. Daggubati, et al. “Cryptography and Reference Sequence Based DNA/RNA Sequence Compression Algorithms,” Journal homepage: http://iieta. org/journals/isi, vol. 27, no. 3, pp. 509–514, 2022.

[6] Chanda, P., et al. "Information Theory in Computational Biology: Where We Stand Today". Entropy, 2020, 22, 627. https://doi.org/10.3390/e22060627

[7] Alyami, Sultan, Chun-Hsi, Huang. "Nongreedy Unbalanced Huffman Tree Compressor for Single and Multifasta Files". Journal of Computational Biology 27. 6(2020): 868-876.

[8] Jan Voges, et al., "GABAC: an arithmetic coding solution for genomic data", Bioinformatics, Volume 36, Issue 7, April 2020, Pages 2275–2277, https://doi.org/10.1093/bioinformatics/btz922.

[9] S. Fiergolla, P. Wolf. "Improving Run Length Encoding by Preprocessing," 2021 Data Compression Conference (DCC), Snowbird, UT, USA, 2021, pp. 341-341, doi: 10.1109/DCC50243.2021.00051.

[10] Y. -L. Tsai and J. -J. Ding. "An Improved LZW Algorithm for Large Data Size and Low Bitwidth per Code," TENCON 2021 - 2021 IEEE Region 10 Conference (TENCON), Auckland, New Zealand, 2021, pp. 203-208, doi: 10.1109/TENCON54134.2021.9707201.

[11] Maël Kerbiriou, Rayan Chikhi. "Parallel decompression of gzip-compressed files and random access to DNA sequences." (2019).

[12] Gulwani, S. “Dimensions in Program Synthesis.” Proceedings of the 12th International ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming, 2010.

[13] Mishra, Ashish & Jagannathan, Suresh. "Specification-Guided Component-Based Synthesis from Effectful Libraries", 2022. 10.48550/arXiv.2209.02752.

[14] Edward Pantridge, Thomas Helmuth. "Solving Novel Program Synthesis Problems with Genetic Programming using Parametric Polymorphism." Proceedings of the Genetic and Evolutionary Computation Conference. ACM, 2023.

[15] Julian Parsert, Elizabeth Polgreen. "Reinforcement Learning for Syntax-Guided Synthesis.", 2023.

[16] Goldberg, Halley, Valentine, Kabanets. "Improved Learning from Kolmogorov Complexity." Proceedings of the Conference on Proceedings of the 38th Computational Complexity Conference. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2023.

[17] Alaskandarani, Kamaludin. "Low Complexity, Low Probability Patterns and Consequences for Algorithmic Probability Applications". Complexity, 2023 : 9696075.

[18] Semkovych, V.; Shymanskyi, V. "Combining OCR Methods to Improve Handwritten Text Recognition with Low System Technical Requirements". Lecture Notes on Data Engineering and Communications Technologies, 2023, pp. 693–702. https://doi.org/ 10.1007/978-3-031-24475-9_56.

[19] Lesia Mochurad. “Optimization of Regression Analysis by Conducting Parallel Calculations”. COLINS-2021: 5th International Conference on Computational Linguistics and Intelligent Systems, April 22–23, 2021, Kharkiv, Ukraine, pp. 982-996.

[20] Zhang, Yihong, et al.. "Better Together: Unifying Datalog and Equality Saturation", 2023.

[21] National Center for Biotechnology Information (NCBI)[Internet]. Bethesda (MD): National Library of Medicine (US), National Center for Biotechnology Information; [2023]. Available from: https://www.ncbi.nlm.nih.gov/

[22] Genome Aggregation Database (gnomAD)[Internet]. (RRID: SCR_014964). Available from: http://gnomad.broadinstitute.org/