An MDL-Based Genetic Algorithm for Genome Sequence Compression

M. Zohaib Nawaz
Shenzhen University
Shenzhen, China
2251271003@email.szu.edu.cn

M. Saqib Nawaz
Shenzhen University
Shenzhen, China
msaqibnawaz@szu.edu.cn

Philippe Fournier-Viger*
Shenzhen University
Shenzhen, China
philfv@szu.edu.cn

Vincent S. Tseng
National Yang Ming Chiao Tung University
Hsinchu, Taiwan
vtseng@cs.nycu.edu.tw

***Abstract*—The exponential growth of genomic data has posed significant challenges for lossless compression of genome sequences. While recent reference-free genome compressors have shown promising results, they often fail to fully leverage the inherent sequential structure of genome sequences, require substantial computational resources and lack (or have limited) interpretability. This paper presents a novel genome compression method that employs the Minimum Description Length (MDL) principle, which is based on the idea that the best model for a given dataset is the one that provides the shortest description of that dataset. The proposed compressor, called GMG (Genetic algorithm for MDL-based Genome compression), integrates a genetic algorithm to identify optimal k-mers (patterns) in a model to best compress the genome data. Experimental results across various datasets demonstrate that GMG outperforms state-of-the-art genome compressors in terms of bits-per-base compression and computational efficiency. Furthermore, it is demonstrated that the optimal patterns identified by GMG for compression can also be utilized for genome classification, offering a multifunctional advantage over previous compressors. GMG is freely available at github.com/MuhammadzohaibNawaz/GMG**

***Index Terms*—Genome sequences, MDL, GA, Crossover, Mutation**

I. Introduction

Genomes, composed of four chemical compounds known as nucleotide bases (adenine (A), cytosine (C), thymine (T), and guanine (G)), represent the complete genetic material of an organism and are generally stored in the FASTA format. They can now be sequenced rapidly and at a low cost, thanks to advanced sequencing technologies, and can be shared on public repositories such as NCBI, NMDC, GISAID, DDBJ and EBI. But this deluge of genomics data [1], [2] presents significant challenges, particularly in terms of efficient compression for storage, management, transmission and processing. Genomes can be compressed with general-purpose compressors designed particularly for text, such as gzip (gzip.org), bzip2 (sourceware.org/bzip2), or LZMA (7-zip.org). However, specialized compressors achieve higher compression than general-purpose ones, as they employ various models that account for the small alphabet size, diverse biological characteristics such as repeats, and high level of substitutions [3], [4].

The development of specialized compression techniques for genomic data started with Biocompress [5]. Since then, numerous methods have been developed for lossless compression of genomic data, which can be categorized into two main types [6]: (1) vertical (or reference-based) methods that focus on finding intra-sequence similarities among sequences. Notable examples include RBFQC [7], LMSRGC [8], ERGC [9], FRESCO [10] and RSS [11]. (2) horizontal (or reference-free) methods that focus on finding inter-sequence similarities among sequences. Examples of these include LEC-Codec [12], GenCoder [13], JARVIS2 [4], GeCo3 [3], NUHT [14], DeepDNA [15], and XM [16]. Reference-based methods generally achieve better compression ratios, especially for genome sequences derived from the same species and with long read lengths. However, they are not self-contained and their success depends on the availability of a good reference genome sequence, which is also required during decompression. Moreover, if the reference genome is not representative of the sequences being analyzed, it can introduce bias and lead to inaccurate results. An important advantage of reference-free genome compression is that some methods can not only reduce data storage but also facilitate genome and metagenomic analysis and classification. For example, the studies [17]–[19] utilized a compression-based feature (Normalized Compression (NC)) and some other features for taxonomic classification.

The introduction and widespread use of the FASTA format have standardized the representation of genomic data (in a visible horizontal range) alongside annotations (headers) [4], with nucleotide or amino acid sequences generally covering the bulk of this data. Various tools, such as MBGC [20], NAF [21], MFCompress [22] and Deliminate [23] utilize specialized compression algorithms in combination with simple header coding. These algorithms predominantly model the presence of exact or approximate repeated and inverted repeated regions through techniques such as bit encoding, context modeling, dictionary approaches and statistical models, including Markov models, run-length encoding and Huffman coding. Recently, there has been a trend toward developing learning-based genome compression methods utilizing neural networks [3], [4], [12], [13], [24], [25]. While these methods achieved better results, they also present several limitations, including high computational complexity, extended running times, limited generalization, lack of interpretability, overfitting issues, and sensitivity to hyperparameters.

* corresponding author.

In this paper, we take a different approach and propose a genome compressor based on the Minimum Description Length (MDL) principle [26]. The MDL principle is founded on the idea that 'the best model compresses the data (in this case, genome sequences) most effectively'. In other words, the optimal model for a given dataset is the one that minimizes the total length of the description of both the model and the data when encoded using that model. The optimal model utilizes a structure called the Code Table (CT), which not only stores all the k-mers (also referred to as patterns) that best represent the data but also defines the encoding scheme for these patterns in the original dataset. However, the optimal patterns of bases in genome sequences are distributed throughout a huge search space, making exhaustive exploration of this space infeasible. The presence of numerous redundant patterns also complicates the process of identifying the truly optimal patterns. To address this challenge, a heuristic approach is proposed, leveraging a Genetic Algorithm [27] for pattern selection, to effectively guide the search for optimal patterns while pruning less promising ones, thereby achieving better compression.

The proposed method, called GMG (GA for MDL-based Genome compression), directs the search toward promising patterns that contribute to improved compression ratios without the need to exhaustively evaluate all possible combinations of patterns. Moreover, the method is not reliant on various costly functions used in the traditional MDL-based approach. Instead, it provides an adaptive and efficient framework for identifying a set of frequent k-mers that offers a superior lossless compression of the database. The results on various datasets demonstrate an improved performance of GMG over state-of-the-art reference-free genome compressors in terms of bits-per-base compression and computational efficiency. Moreover, it is shown that the optimal k-mers found by GMG can be used for genome classification.

The next sections of this paper are organized as follows: Section 2 describes how the MDL principle can be applied for genome sequence compression. Section 3 describes the proposed GMG compressor for genomes. Then, Section 4 presents the experimental evaluation of GMG against state-of-the-art genome sequence compressors. Finally, Section 5 provides the conclusion.

II. Adapting the MDL Principle for Genome Sequence Compression

The MDL principle is a general method for model selection that aims to find the model that best describes some data. This section explains its adaptation to genome compression and then discusses the potential of using a GA as a heuristic to obtain better compression.

Let M represent a model that is used to describe a database D. Let $L(D|M)$ represent the compression size, in bits, of D when encoded with the model M and $L(M)$ represent M's description size in bits. The MDL [26] is defined as follows:

For a set of models $\mathbb{M}$, the best model $M \in \mathbb{M}$ is the one that minimizes the compression size, given by:

$$L(D,M) = L(D|M) + L(M) \tag{1}$$

To utilize the MDL principle for genome compression, it is necessary to define the structure of a model $M \in \mathbb{M}$, its application to describe a database, as well as the model's encoding in bits.

Let NB represent the set of nucleotide bases (A, C, G, T). A database D in this context is a list of genome sequences over items in NB. A sequence $s \in \mathcal{P}$ (NB) is an ordered arrangement of bases, referred to as a baseset. A baseset is equivalent to a k-mer (denoted as kM), which is a contiguous sequence of k bases from NB.

A model is a set of k-mers $M = \{s_1, s_2, ..., s_m\}$ used for compressing a database D, and is represented by a structure called a *code table*, denoted as CT. This latter is a two-column dictionary where the left column contains k-mers and the right column provides a code for each k-mer. The CT comprises non-zero usage k-mers, ordered first by descending length, then by decreasing support (the number of sequences in D that contain a k-mer), and finally in ascending lexicographical order. This defines a total order on k-mers called the *Standard Cover Order*. The lengths of the codes in the right column are important to assign shorter codes to frequent k-mers to achieve greater compression.

A code table $CT \subseteq \{(kM, code(kM)) \mid kM \in CS\}$, represents a compressing model, where a code $code(kM)$ is assigned to each k-mer $kM \in CS$. The coding set (CS) is the set of all k-mers from the model. A *cover function*, denoted as $cover(s)$, is defined to encode each sequence s from D over NB with a CT. The function *cover(s)* identifies the set of patterns from CS that, when applied, encode a sequence s. For a k-mer $kM \in CT$, its *usage(kM)* is the count of sequences $s \in D$ having kM in their cover [28]. To determine the optimal CT using the MDL principle, it is necessary to consider both the compressed size of D and CT. To achieve the optimal compression of D, codes in CT that are more frequent should be assigned shorter lengths. This can be achieved by using the Shannon-entropy to calculate the optimal code length for kM.

$$L(kM|D) = -log_2(P(kM|D) \tag{2}$$

where

$$P(kM|D) = \frac{usage(kM)}{\sum_{kM \in CT} \text{usage}(kM)} \tag{3}$$

The length of the encoding of s is the summation of the code lengths of the k-mers in its cover.

$$L(s|CT) = \sum_{kM \in cover(s)} L(kM|CT) \tag{4}$$

Now, the size of the encoded D is the sum of the sizes of the encoded sequences.

$$L(D|CT) = \sum_{s \in D} L(s|CT) \tag{5}$$

For the size of the CT, only those k-mers are considered that have a non-zero usage. The encoded size of the CT is:

$$L(CT|D) = \sum_{\substack{kM \in CT \\ usage(kM) \neq 0}} L(kM|CT) \tag{6}$$

$L(kM|CT)$ is the summation of all the different code lengths. The optimal set of k-mers are the one whose associated CT table minimizes the total encoded size:

$$L(D, CT) = L(CT|D) + L(D|CT) \tag{7}$$

To compress genome sequences from a database D, our goal is to find the sets of k-mers that best describe D. Using $L(D, CT)$, we formalize this goal using the MDL as:

Minimal Ordered k-mer Set Problem: Let NB be a set of bases, D be a database over NB, *cover* be a cover function and kM a collection of candidate k-mers ($kM \in \mathbb{P}(NB)$). The problem is to determine the smallest set of patterns $P \in kM$ in a way that the overall compressed size, $L(CT, D)$, is minimal for the corresponding code table CT. Note that the set of k-mers in a CT, i.e. $\{kM \in CT\}$ represents the CS.

Using MDL for compression thus requires finding an optimal set of k-mers with their codes to build a CT, but the search space of possible models is too large to be searched exhaustively. Hence, GA is utilized as heuristic in GMG to select some initial k-mers and then iteratively evolve them by using two operations: crossover and mutation. The fitness function is $L(CT|D)$. More details are presented in the next section

III. The GMG compressor

This section introduces the proposed GMG algorithm to discover a representative set of k-mers to effectively compress genome sequences.

The flowchart of GMG, highlighting its main steps, is shown in Fig. 1. The algorithm begins by generating an initial population of k-mers, representing potential solutions for compressing the dataset D. The generated k-mers are then evolved through an iterative GA process that involves the following steps:

1) **Selection**: k-mers or sequences from the population, referred to as parents, are randomly selected (generated) to undergo crossover and mutation operations.
2) **Crossover**: The selected sequences (called parents) undergo crossover, wherein sub-sequences in the parents are combined to produce new sequences (called children), mimicking genetic recombination to explore new areas of the solution space.
3) **Mutation**: Random changes are introduced in the child sequence, obtained after crossover operation. This modification enhances diversity within the population.
4) **Evaluation**: The fitness of the k-mers resulting from crossover and mutation is assessed by measuring their occurrence frequency in the dataset. Higher occurrence frequency sub-sequences are assigned lower codes, leading to improved compression of the dataset.
5) **Stopping Criteria**: GMG terminates when a predefined condition is met, that is adding k-mers in the CT till achieving a satisfactory compression ratio.

Algorithm III presents the pseudo-code for GMG. The algorithm operates with two inputs: a genome sequence dataset D and a threshold, called *maxCTSize*, which defines the maximum size of the CT, which acts as a termination condition ensuring that the algorithm stops once the *CT* reaches the specified size. This provides a practical limit to the search space and reduces computation time. The *maxCTSize* also offers flexibility to users, enabling them to restrict the number of output k-mers.

The output of GMG is a *CT* containing representative k-mers that efficiently compress the dataset. Additionally, GMG can be configured to run without the *maxCTSize* parameter, making it fully parameter-free. In this configuration, the algorithm terminates when the compression ratio stabilizes, specifically, if the ratio does not improve over a set number of generations.

[H] GMG [1] Genome sequence dataset D, $maxCTSize$ CT that best compresses the dataset $population \leftarrow \{A, C, T, G\}$ $CT \leftarrow \emptyset$ len(CT) $\neq$ $maxCTSize$ $P_1, P_2 \leftarrow$ Two random unique k-mers (length [2,6])$Pocc_1 \leftarrow$ CountOccurrences(P_1, D) $Pocc_2 \leftarrow$ CountOccurrences(P_2, D) $C_1, C_2 \leftarrow$ Crossover(P_1, P_2) $C_1 \leftarrow$ Mutation($C_1, population$) $C_2 \leftarrow$ Mutation($C_2, population$) $Random() > 0.5$ $Cocc_1 \leftarrow$ CountOccurrences(C_1, D) $Cocc_1 > Pocc_1$ $P_1 \leftarrow C_1$ $Pocc_1 \leftarrow Cocc_1$ $Cocc_2 \leftarrow$ CountOccurrences(C_2, D) $Cocc_2 > Pocc_2$ $P_2 \leftarrow C_2$ $Pocc_2 \leftarrow Cocc_2$ (max. number of generations is attained) $CT \leftarrow CT \cup \{P_1, P_2\}$ Replace P_1, P_2 in D with codes
Return CT

GMG starts by initializing a population with four nucleotides (line 1). The CT is initialized as empty (line 2), and a while loop runs until the size of CT reaches the specified $maxCTSize$ threshold. In each iteration, two random k-mers P_1 and P_2 with k lengths between 2 and 6 are created (line 4), and their occurrences in the dataset are counted (lines 5, 6). Crossover (line 8) and mutation (lines 9, 10) operators are applied to evolve these k-mers. The occurrences of the resulting k-mers (C_1 or C_2) are then recalculated (lines 11-23) and compared to the original ones. If the evolved k-mers have more occurrences, they replace the parent k-mers (lines 14, 20). This process continues over multiple generations, refining the result, and at the end of the generations, the evolved parents are added to the *CT* (line 25). The k-mers in the CT are assigned (based on occurrences) codes that are used for encoding in the dataset D (line 26). The final output of GMG is the CT—a set of sequences that optimally compresses the dataset.

The process of calculating the occurrences for both C_1 and C_2 can be time-consuming, particularly with large datasets. To address this, we introduced a probabilistic approach to improve efficiency (line 11). Instead of counting occurrences for both

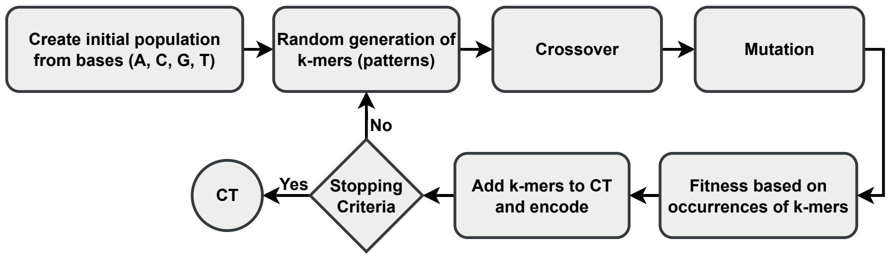
Create initial population from bases (A, C, G, T)
Random generation of k-mers (patterns)
Crossover
Mutation
Fitness based on occurrences of k-mers
Add k-mers to CT and encode
Stopping Criteria
No
Yes
CT

Fig. 1. Flowchart of the GMG genome sequence compressor.

sequences, we assigned a 50% chance for either C_1 or C_2 to be selected, ensuring only one sequence is evaluated per iteration. This significantly reduces the computational cost. Our experiments showed that this approach accelerates the algorithm without compromising compression quality.

A key feature of GMG is its ability to maintain or improve solution quality over iterations. As outlined in Algorithm III (lines 13-16, 19-22), the compression ratio—our main optimization metric—either improves or stays constant, never worsening. GMG achieves this through a selective update mechanism: if a child k-mer offers a better compression ratio than its parent, the parent is replaced by the child. If not, the parent remains unchanged. This ensures that solution quality does not degrade over time. This phenomenon is discussed further in detail in the result section (see convergence results in section IV-C3).

```
[!ht] Crossover [1] $PkM_1, PkM_2$: Two parent k-mers $CkM_1, CkM_2$: Two child k-mers
procedure SPC($PkM_1, PkM_2$)
    $s \leftarrow$ min(len($PkM_1$), len($PkM_2$))
    $cp \leftarrow$ randomInt(1, $s$)    // $1 \leq cp \leq s$
    $CkM_1 \leftarrow PkM_1[1, cp] \parallel PkM_2[cp + 1,$ len($PkM_2$)$]$
    $CkM_2 \leftarrow PkM_2[1, cp] \parallel PkM_1[cp + 1,$ len($PkM_1$)$]$
    Return $CkM_1$ and $CkM_2$
end procedure
```

Algorithm III, and III provide the pseudo-code for the crossover and mutation operators, respectively. In the crossover operator [29], a random crossover point (*cp*) is selected, and the sequences to the left (or right) of *cp* are exchanged between the two parent k-mers to produce two new child k-mers. The symbol $\parallel$ represents the concatenation of sub k-mer. A simple example is provided next to explain the single-point crossover operators. Let PkM_1 and PkM_2 be two k-mers:

$$PkM_1 = \textit{AGTCGT}$$
$$PkM_2 = \textit{TGCGA}$$

Let n represent the length of the largest k-mer. A random *cp* ($(1 \leq cp \leq n$) is randomly selected. For *cp* = 3, the single point crossover generates the two child k-mers as:

$$CkM_1 = \textit{AGTGA}$$
$$CkM_2 = \textit{TGCCGT}$$

The mutation operation [30] follows the crossover operation and is applied independently to k-mers. The mutation is applied probabilistically, selecting either CkM_1 or CkM_2 with a 50/50 probability, as described in Algorithm III. It introduces randomness into the search process, helping to avoid premature convergence to local optima. During mutation, a selected NB in the k-mer is modified by replacing it with a randomly chosen NB from the population. For example, a mutation of CkM_1 and CkM_2 can be:

$$CkM_1' = \textit{ACTGA}$$
$$CkM_2' = \textit{TGCTGT}$$

```
[!ht] Standard Mutation [1] $kM$: A k-mer and $population$: set of all items A mutated k-mer
procedure SM($kM$)
    $i \leftarrow$ randomInt(1, len($kM$))
    $alter \leftarrow$ randomSelect($population$, 1)    // 1-mer
    $kM[i] \leftarrow alter$    // $kM[i] \neq alter$
    Return $kM$
end procedure
```

The GMG algorithm terminates when the stopping condition of the GA is met, specifically when the number of sequences in the CT equals the user-defined *maxCTSize* threshold. This threshold can be adjusted by the user based on their desired output size.

IV. Experimental Evaluation

In this section, the performance of GMG is benchmarked against two state-of-the-art reference-free compression methods: GeCo3 [3], and JARVIS2 [4]. GeCo3 integrates neural networks with specific DNA compression models, using a mixture of multiple context and substitution-tolerant context models to achieve improved compression ratios across various datasets. Similarly, JARVIS2 employs a mixture of multiple finite-context models (FCMs) and weighted local stochastic repeat models (WLSRMs), enhanced by a neural network and arithmetic encoding to optimize compression efficiency while minimizing RAM usage.

Next, we provide the overview of the system configuration utilized for the experiments, followed by a comprehensive description of the datasets and benchmarking results.

A. System Configuration

The experiments were conducted on a desktop computer featuring a 12th Gen Intel® Core™ i7-12700 processor (2.10 GHz), running Windows 11 for Education in performance mode. The machine was equipped with 16 GB of RAM and 1 TB of disk storage, an adequate configuration for handling the large datasets involved in the experiments.

B. Datasets

The benchmarking of GMG against GeCo3 and JARVIS2 includes 2 datasets.

- **Dataset 1 (DS1)**: A comprehensive balanced dataset [31] composed of 15 genomic sequences from a wide array of species, including (1) chromosome 2 of *Gallus gallus* (GaGa), (2) chromosome 3 of *Danio rerio* (DaRe), (3) chromosome 1 of *Oryza sativa Japonica* (OrSa), (4) chromosome 2 of *Drosophila miranda* (DrMe), (5) chromosome 4 of the *reference human genome* (HoSA), (6) genome of *Entamoeba invadens* (EnIn), (7) genome of *Schizosaccharomyces pombe* (ScPo), (8) genome of *Plasmodium falciparum* (PlFa), (9) genome of *Escherichia coli* (EsCo), (10) genome of *Haloarcula hispanica* (HaHi), (11) genome of *Aeropyrum camini* (AeCa), (12) genome of *Helicobacter pylori* (HePy), (13) genome of *Yellowstone Lake mimivirus* (YeMi), (14) genome of *Aggregatibacter phage S1249* (AgPh), and (15) genome of *Bundibugyo ebolavirus* (BuEb). This diverse dataset ensures a robust evaluation across various organisms, representing a range of genomic characteristics.
- **Dataset 2 (DS2)**: The human genome T2T sequence (Chm13 version 2.0) [32], which consists of 3,117,292,120 bases. This dataset, representing the concatenated sequence of the whole human chromosomes, serves as a substantial benchmark to assess compression efficiency on large, real-world data. Assuming a uniform distribution of symbols, the baseline representation of this sequence without compression is approximately 779,323,030 bytes.

C. Benchmarking Results

First, the results for the compression ratio are discussed, followed by the results for compression time (in seconds). The convergence behavior of GMG is then analyzed to observe how it systematically refines the compression ratio over successive iterations for optimal k-mer selection. Finally, the quality of the optimal k-mers in the *CT* is evaluated by utilizing them as features for classification. The details are as follows:

1) Compression Ratio (BPB): The benchmark results for BPB on both datasets are presented in Table 1. In nearly every case, GMG outperforms GeCo3 and JARVIS2 in terms of compression ratio.

For DS1, across the various genomic sequences, GMG consistently achieves a lower BPB value (indicating better compression) compared to the other two compressors. For instance, on GaGa, GMG achieves a BPB of 1.56, outperforming GeCo3's 1.85 and JARVIS2's 1.79. Similarly, for EsCo, GMG achieves a BPB of 1.73, which is lower than GeCo3's 1.93 and JARVIS2's 1.88. Overall, on average, GMG achieves a compression ratio of 1.59 BPB, compared to GeCo3's 1.83 BPB and JARVIS2's 1.77 BPB in DS1. This translates to GMG providing 15.1% better compression than GeCo3 and 11.3% better than JARVIS2. Importantly, all these compression results are achieved using only 4 k-mers

TABLE I
BENCHMARK RESULTS FOR COMPRESSION (IN BPB)

Dataset↓, Method→		**GeCo3**	**JARVIS2**	**GMG**
15***DS1**	HoSa	1.69	**1.58**	**1.58**
	GaGa	1.85	1.79	**1.56**
	DaRe	1.65	1.50	**1.48**
	OrSa	1.70	1.52	**1.66**
	DrMe	1.88	1.82	**1.58**
	EnIn	1.73	1.52	**1.55**
	ScPo	1.90	1.88	**1.57**
	PlFa	1.74	1.89	**1.61**
	EsCo	1.93	1.88	**1.73**
	HaHi	1.89	1.83	**1.56**
	AeCa	1.94	1.90	**1.59**
	HePy	1.85	1.78	**1.69**
	YeMi	1.84	1.83	**1.55**
	AgPh	1.96	1.96	**1.53**
	BuEb	1.98	1.98	**1.65**
1***DS2**	Human Genome	1.42	**1.39**	1.40
Average of DS1		1.83	1.77	**1.59**
Overall average		1.80	1.75	**1.57**

in the *CT*, which is one of the inputs to GMG, specifically *maxCTSize*. These results clearly demonstrate GMG's superior compression effectiveness.

For DS2 (the human genome), GMG compresses the 3.1 billion DNA symbols to a BPB of 1.40, while GeCo3 and JARVIS2 achieve BPBs of 1.42 and 1.39, respectively. This compression is comparable to JARVIS2 but slightly less effective than Geco3. Notably, for this dataset, the number of k-mers in the *CT* is 8, as opposed to 4 for DS1. This increase is necessary due to the dataset's large size, requiring more k-mers to achieve a compression ratio comparable to the other two methods. However, the overall average compression results in Table 1 (for both datasets) demonstrates that GMG maintains a strong performance, achieving an average compression ratio of 1.57 BPB. This represents an improvement of 14.6% over GeCo3 (1.80 BPB) and 11.4% over JARVIS2 (1.75 BPB).

2) Compression Time: In addition to superior compression ratios, GMG is significantly faster in terms of compression time. For example for sequence GaGa in DS1, GMG compresses the data in 58 seconds, whereas GeCo3 takes 74.01 seconds and JARVIS2 takes 354 seconds. Similarly, for EsCo, GMG completes the compression in 1.50 seconds, compared to GeCo3's 2.10 seconds and JARVIS2's 9.01 seconds. Additionally, GMG achieves an average compression time of 16.45 seconds, while GeCo3 takes 26.29 seconds and JARVIS2 takes 111.85 seconds on DS1. This means GMG is 1.5 times faster than GeCo3 and more than 6.7 times faster than JARVIS2 on average for DS1.

For the human genome (DS2), the compression time advantage of GMG becomes even more pronounced. GMG compresses the entire dataset in approximately 1,200 seconds, while GeCo3 takes 30,000 seconds and JARVIS2 takes 25,800 seconds. This means GMG is 25 times faster than GeCo3 and 21.5 times faster than JARVIS2 for the human genome sequence. When looking at the overall average compression times in Table 2 (for both datasets), GMG continues to outperform with an average time of 90.42 seconds, compared

TABLE II
BENCHMARKS RESULTS FOR COMPRESSION TIME (IN SECONDS)

Dataset↓, Method→		GeCo3	JARVIS2	GMG
15***DS1**	HoSa	102.10	540	**76.20**
	GaGa	74.01	354	**58.00**
	DaRe	110.87	460.01	**55.64**
	OrSa	32.74	130.78	**17.46**
	DrMe	25.92	67.68	**13.17**
	EnIn	21.01	59.25	**11.33**
	ScPo	10.42	23.75	**4.48**
	PlFa	7.94	18.55	**4.13**
	EsCo	2.10	9.01	**1.50**
	HaHi	2.51	7.80	**2.10**
	AeCa	2.57	3.64	**1.38**
	HePy	1.90	2.51	**1.15**
	YeMi	0.18	0.65	**0.15**
	AgPh	0.17	0.15	**0.10**
	BuEb	**0.03**	0.08	**0.03**
1***DS2**	Human Genome	30,000	25,800	**1,200**
Average of DS1		26.29	111.85	**16.45**
Overall average		1,899.65	1,717.36	**90.42**

to 1,899.65 seconds for GeCo3 and 1,717.36 seconds for JARVIS2. This demonstrates that GMG is over 21 times faster than GeCo3 and 18.9 times faster than JARVIS2 on average on both datasets.

3) Convergence: Another experiment was conducted on the two datasets to observe the convergence behavior of GMG, specifically how it systematically refines the compression ratio over successive iterations to improve k-mer selection for optimal compression. The convergence of the compression ratio was evaluated based on two factors: the number of k-mers added to the CT and the number of generations used to evolve a set of k-mers before they are incorporated into the CT.

The results are shown in Fig. 2. The bottom x-axis represents the number of k-mers that have been added to the *CT*, while the top x-axis indicates the number of generations that GMG utilizes to evolve each set of k-mers. The y-axis reflects the compression ratio, which quantifies the algorithm's efficiency in compressing the data. For clarity, the number of sequences and generations are observed in the range of [1,4] and [1,10], respectively.

In the figure, the compression ratios of several genome sequences in two datasets exhibit a common pattern: an initial high compression ratio that declines as more k-mers are added. This suggests that the first k-mer added to the CT contributes significantly to improving compression, while subsequent k-mers lead to more modest gains. For example, the AeCa and HePy in DS1 show a gradual reduction in the compression ratio as k-mers and generations increase, stabilizing around the 10th generation. Similarly, the sequences DaRe and OrSa follow similar trends, but with a steeper decline at the end, indicating that these datasets benefit from the later optimal k-mers, which enhance compression efficiency more effectively. The EsCo dataset demonstrates more stable behavior, with only slight declines throughout the compression process, highlighting that their compression does not change significantly with more generations. It is important to note that the Human Genome in DS2 utilized more k-mers than the other sequences in DS1, employing 8 k-mers compared to the 4 k-mers used for the rest. This means that this sequence required more k-mers compared to other sequences for better representation and compression.

4) Classification: We evaluate the quality of optimal k-mers in the CT by utilizing them as features for training classification. The classification is performed using Weka [33], where six well-known classifiers were trained using the k-mers as features: Naive Bayes (NB), Logistic Regression (LR), Support Vector Machine (SVM), k-nearest Neighbor (kNN), Decision Tree (DT) and Random Forest (RF). Note that the classifiers were evaluated using their default hyperparameters, along with 10-fold cross-validation. The performance of classifiers was evaluated using the accuracy metric. In our case, accuracy is defined as the percentage of correctly classified instances based on the optimal k-mers derived from GMG. The resulting accuracies achieved by the classifiers are presented in Table III.

TABLE III
CLASSIFIERS ACCURACY (IN %) ON OPTIMAL K-MERS

Dataset↓, Classifier→		NB	LR	SVM	kNN	DT	RF
15***DS1**	HoSa	85.29	89.70	**94.11**	88.23	**94.11**	92.64
	GaGa	60.29	94.11	**94.11**	85.29	**94.11**	92.64
	DaRe	92.64	**95.58**	94.11	91.17	94.11	**95.58**
	OrSa	50	92.64	92.71	86.76	**94.85**	89.70
	DrMe	89.70	88.23	**94.11**	89.70	92.03	91.17
	EnIn	83.82	88.23	93.42	88.23	92.64	**93.75**
	ScPo	89.70	92.64	**92.82**	85.29	90.36	89.62
	PlFa	77.94	89.70	**94.88**	88.23	92.61	89.70
	EsCo	92.64	89.70	90.78	89.70	**94.73**	94.11
	HaHi	79.41	91.17	92.17	89.70	91.26	**92.85**
	AeCa	89.70	92.64	**92.75**	86.76	92.42	92.64
	HePy	91.17	**94.11**	93.63	85.29	90.97	93.46
	YeMi	88.23	92.64	**94.11**	88.23	93.75	93.97
	AgPh	70.58	88.23	**94.11**	80.88	**94.11**	88.23
	BuEb	92.64	95.58	94.11	91.17	94.11	**95.58**
1***DS2**	Human Genome	82.35	83.82	**88.23**	76.47	**88.23**	83.82
Overall average		82.25	91.17	**93.13**	86.94	92.77	91.84

SVM achieved 93.13% accuracy on average, followed by DT (92.77%), RF (91.84%) and LR (91.17%). By comparison, the studies [17]–[19] used compression measures (such as NC) with some other features for the taxonomic classification of viruses, Archaea and metagenomic sequences, respectively. The XGBoost classifier performed best on the provided features and achieved an average accuracy of 82.15% [17], 92.10% [18] and 95% [19]. Here in this paper, the optimal k-mers in the CT enhance the interpretability of GMG by revealing the distribution and frequency of k-mers that contribute significantly to the compression of genome sequences. Moreover, optimal k-mers can not only be used as features in the classification process, but they also offer the opportunity to develop a specialized classifier based on the CTs.

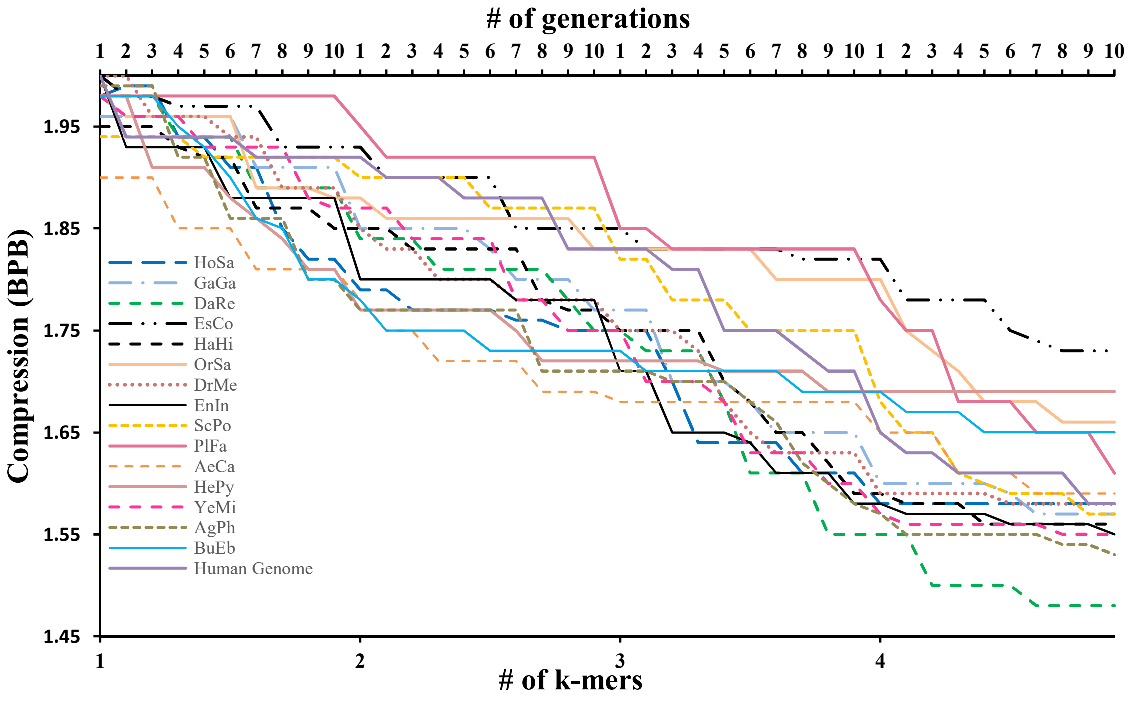

Fig. 2. Comparison of compression with respect to number of k-mers and generations

In summary, our benchmark results clearly demonstrate that GMG significantly outperforms state-of-the-art reference-free compression tools, GeCo3 and JARVIS2, both in terms of compression efficiency (BPB) and speed. Moreover, the efficiency of GMG as a descriptor of data (genome sequences) is evident in the initial classification results. These findings establish GMG as an optimal solution for handling genomic data, offering advantages not only in terms of data storage and processing speed but also facilitate genome classification, making it suitable for a wide range of applications in genomic research and bioinformatics workflows.

V. Conclusion

In this paper, we have introduced GMG, a novel method that combines a GA with the MDL principle for efficient reference-free lossless compression of genomic data. By leveraging GA to identify optimal k-mers, GMG constructs a model that effectively captures the sequential structure of genome sequences for better compression performance. GMG is benchmarked against the state-of-the-art reference-free compression methods, GeCo3 and JARVIS2, across two genomic datasets. The results demonstrated that GMG achieved superior performance, as evident by lower BPB values, significantly reduced compression times and provided high accuracy results when optimal k-mers, derived by GMG, were used for classification. The MDL principle ensures that GMG maintains an optimal balance, neither being overly complex nor simplistic, by aligning its complexity with its ability to accurately describe or (represent) the genome data. GMG has the potential to play an important role in the ever-expanding field of genomic data analysis, offering more cost-effective storage solutions and enabling faster, more streamlined workflows in research and clinical environments.

While GMG demonstrated superior performance, future enhancements in parallel processing, multithreading, or GPU acceleration could make it even faster, particularly when dealing with large-scale datasets. Another future direction is to evaluate the scalability of the developed method using additional genomic and metagenomic datasets. There is also potential to further improve compression ratio by allowing 'gaps' in between nucleotide bases. Lastly, another direction is the development of a specialized classifier based on CTs that stores optimal k-mers.

References

[1] G. Berger, J. Peng, M. Singh, "Computational solutions for omics data", Nat. Rev. Genet. vol. 14, no. 5, pp. 333–346, 2013.

[2] M. S. Nawaz, M. Z. Nawaz, Z. Junyi, P. Fournier-Viger, J.-F. Qu, "Exploiting the sequential nature of genomic data for improved analysis and identification", Comput. Biol. Med. vol. 183, 109307, 2024.

[3] M. Silva, D. Pratas, A. J. Pinho, "Efficient DNA sequence compression with neural networks", GigaScience, vol. 9, no. 11, giaa119, 2020.

[4] D. Pratas, A. J. Pinho, "JARVIS2: a data compressor for large genome sequences", in Proc. of DCC, 2023, pp. 288–297.

[5] S. Grumbach, F. Tahi, "Compression of DNA sequences", in Proc. DCC, 1993, pp. 340–350.

[6] M. Hosseini, D. Pratas, A. J. Pinho, "A survey on data compression methods for biological sequences", Information, vol, 7, no. 4, 56, 2016.

[7] S. Kumar, M. P. Singh, S. R. Nayak, et al, "A new efficient referential genome compression technique for FastQ files", Funct. Integr. Genomics. vol. 23, no. 333, 2023.

[8] Z. Lu, L. Guo, J. Chen. "Reference-based genome compression using the longest matched substrings with parallelization consideration", BMC Bioinform. vol. 24, no. 369, 2023.

[9] S. Saha, S. Rajasekaran, "ERGC: an efficient referential genome compression algorithm, Bioinforma. vol. 31, no. 21, pp. 3468–3475, 2015

[10] S. Wandelt, U. Leser, “FRESCO: referential compression of highly similar sequences”, IEEE/ACM Trans. Comput. Biol. Bioinform. vol. 10, no. 5, pp. 1275–1288, 2013.
[11] K. -O. Cheng, N. -F Law, W. -C Siu, “Clustering-based compression for population DNA sequences”, IEEE/ACM Trans. Comput. Biol. Bioinform. vol. 16, no. 1, pp. 208–221, 2019.
[12] Z. Sun, Z., M. Wang, S. Kwong, “LEC-Codec: Learning-based genome data compression”, IEEE/ACM Trans. Comput. Biol. Bioinform. pp. 1–12, 2024.
[13] S. K. Sheena, M. S. Nair, “GenCoder: A novel convolutional neural network based autoencoder for genomic sequence”, IEEE/ACM Trans. Comput. Biol. Bioinform. vol. 21, no. 3, pp. 405–415, 2024.
[14] S. Alyami, C. -H Huang, “Nongreedy unbalanced Huffman tree compressor for single and multifasta files”, J. Comput. Bio. vol. 27, no. 6, pp. 868–876, 2020.
[15] R. Wang., Y. Bai, Y. -S Chu, Z. Wang, V. Wang, M. Sun, “DeepDNA: a hybrid convolutional and recurrent neural network for compressing human mitochondrial genomes”, in Proc BIBM, 2018, pp. 270–274.
[16] M. D. Cao, T. I. Dix, L. Allison, C. Mears, “A simple statistical algorithm for biological sequence compression”, in: Proc DCC, 2007, pp. 43–52.
[17] J. M. Silva, D. Pratas, T. Caetano, S. Matos, “The complexity landscape of viral genomes”, GigaScience, vol. 11, giac079, 2022.
[18] J. M. Silva, D. Pratas, T. Caetano, S. Matos, “Feature-based classification of Archaeal sequences using compression-based methods”, in Proc. IbPRIA, 2022.
[19] J. M. Silva, J. R. Almeida, “Enhancing metagenomic classification with compression-based features”, Artif. Intell. Med. vol, 156, 102948, 2024.
[20] S. Grabowski, T. M. Kowalski, “MBGC: multiple bacteria genome compressor”, GigaScience, vol. 11, giab099, 2022.
[21] K. Kryukov, M. T. Ueda, S. Nakagawa, T. Imanishi, “Nucleotide archival format (NAF) enables efficient lossless reference-free compression of DNA sequences”, Bioinfor. vol. 35, no. 19, pp. 3826–3828, 2019.
[22] A. J. Pinho, D. Pratas, “MFCompress: a compression tool for FASTA and multi-FASTA data”, Bioinfor. vol. 30, no. 1, pp. 117–118, 2014.
[23] M. H. Mohammed, A. Dutta, T. Bose, S. Chadaram, S. S. Mande, “DELIMINATE - A fast and efficient method for loss-less compression of genomic sequences: sequence analysis”, Bioinfor. vol. 28, no. 19, pp. 2527–2529, 2012.
[24] K. S. Sheena, M. S. Nair, “DNACoder: a CNN-LSTM attention-based network for genomic sequence data compression”, Neural Comput. Appl. vol. 36, pp. 18363–18376, 2024.
[25] W. Cui, Z. Yu, Z., Liu, G. Wang, X. Liu, “Compressing genomic sequences by using deep learning”, in Proc. ICANN, pp. 92–104, 2020.
[26] P. D. Grunwald, The Minimum Description Length Principle. MIT Press, 2007.
[27] J. H. Holland, Adaptation in natural and artificial systems, MIT Press, 1975.
[28] J. Vreeken, M. van Leeuwen, A. Siebes, “KRIMP: mining itemsets that compress”, Data Min. Knowl. Disc. vol. 23, pp. 169–214, 20011.
[29] M. S. Nawaz, M. Z. Nawaz, O. Hasan, P. Fournier-Viger, M. Sun, “An evolutionary/heuristic-based proof searching framework for interactive theorem prover”, Appl. Soft Comput. 104, 107200, 2021.
[30] M. S. Nawaz, M. Z. Nawaz, O. Hasan, P. Fournier-Viger, M. Sun, “Proof searching and prediction in HOL4 with evolutionary/heuristic and deep learning techniques”, Appl. Intell. vol. 51, pp. 1580–1601, 2021.
[31] D. Pratas, A. J. Pinho, “A DNA sequence corpus for compression benchmark”, in Proc. of PACBB, 2018, pp. 208-215.
[32] S. Nurk, S. Koren, A. Rhie, et al, “The complete sequence of a human genome”, Science, vol. 376, no. 6588, pp. 44–53, 2022.
[33] E. Frank, M. A. Hall, I. H. Witten, ”The WEKA Workbench”, Online Appendix for ”Data Mining: Practical Machine Learning Tools and Techniques”, 4th Edition, Morgan Kaufmann, 2016.