Preprint of:
Nawaz, M. Z., Nawaz, M. S., Fournier-Viger, P., Nawaz, S., Lin, J. C.-W., Tseng, V.S. (2025). Efficient Genome Sequence Compression via the Fusion of MDL-Based Heuristics. Information Fusion. 120: 103083.

Efficient Genome Sequence Compression via the Fusion of MDL-Based Heuristics

M. Zohaib Nawaz[a,b], M. Saqib Nawaz[a], Philippe Fournier-Viger[a,*], Shoaib Nawaz[c], Jerry Chun-Wei Lin[d], Vincent S. Tseng[e]

[a] *College of Computer Science and Software Engineering, Shenzhen University, China*
[b] *Faculty of Computing and Information Technology, Department of Computer Science, University of Sargodha, Pakistan*
[c] *Department of Pharmacy, The University of Lahore, Sargodha Campus, Pakistan*
[d] *Western Norway University of Applied Sciences Bergen, Norway*
[e] *National Yang Ming Chiao Tung University Hsinchu, Taiwan*

Abstract

Developing novel methods for the efficient and lossless compression of genome sequences has become a pressing issue in bioinformatics due to the rapidly increasing volume of genomic data. Although recent reference-free genome compressors have shown potential, they often require substantial computational resources, lack interpretability, and fail to fully utilize the inherent sequential characteristics of genome sequences. To overcome these limitations, this paper presents HMG (Heuristic-driven MDL-based Genome sequence compressor), a novel compressor based on the Minimum Description Length (MDL) principle. HMG is designed to identify the optimal set of k-mers (patterns) for the maximal compression of a dataset. By fusing heuristic algorithms—specifically the Genetic Algorithm and Simulated Annealing—with the MDL framework, HMG effectively navigates the extensive search space of k-mer patterns. An experimental comparison with state-of-the-art genome compressors shows that HMG is fast, and achieves a low bit-per-base. Furthermore, the optimal k-mers derived by HMG for compression are employed for genome classification, thereby offering multifunctional advantages over previous genome compressors. HMG is available at github.com/MuhammadzohaibNawaz/HMG.

Keywords: Genome sequences, Minimum description length, Genetic algorithm, Simulated annealing, Crossover, Mutation

1. Introduction

Genomes, composed of four nucleotide bases—Adenine (A), Cytosine (C), Thymine (T), and Guanine (G)—represent an organism's complete genetic material. Found in diverse environments, such as uranium mines [1], marine habitats [2], ancient cadavers [3], and deep subterranean locations [4], genomes adapt to various conditions, resulting in significant diversity [5]. This diversity includes high copy numbers, heterogeneity, substitution mutations, and structural

*Corresponding author
Email address: philfv@szu.edu.cn (Philippe Fournier-Viger)

rearrangements like fusions, fissions, and inverted repeats [6]. External factors, such as contamination [7], environmental conditions [8], and pathogenic species [9], further complicate genomic data, requiring models capable of handling dynamic, heterogeneous, and potentially imperfect data [10].

Advancements in sequencing technologies like Next-Generation Sequencing (NGS) and Third-Generation Sequencing (TGS) have enabled rapid genome sequencing [11], generating vast amounts of data shared through public repositories such as NCBI (ncbi.nlm.nih.gov), NMDC (nmdc.cn), GISAID (gisaid.org), DDBJ (ddbj.nig.ac.jp), and EBI (ebi.ac.uk). By 2025, genomic data storage needs are projected to reach 2–40 exabytes (EB) annually [12]. This deluge of genomics data [13, 14], coupled with its inherent complexity and diversity, poses challenges for efficient compression, storage, and processing. General-purpose compressors like GZIP (gzip.org), BZIP2 (sourceware.org/bzip2), or LZMA (7-zip.org) can be applied; however, they fail to exploit the genomic characteristics effectively [15]. Specialized compressors, on the other hand, achieve higher compression ratios by leveraging genomic-specific models tailored to features like repeats and substitutions [16].

Since the introduction of Biocompress [17], the first specialized genome compressor in 1993, numerous lossless genomic compression methods have been developed, which can be categorized into two types [18]: (1) vertical (reference-based) methods, which identify intra-sequence similarities by exploiting structural (e.g., palindromes) and statistical properties. Notable examples include RBFQC [19], LMSRGC [20], ERGC [21], FRESCO [22], and RSS [23]; and (2) horizontal (reference-free) methods, which focus on inter-sequence similarities between target and reference sequences. Examples include GraSS [24], JARVIS3 [25], LEC-Codec [26], GenCoder [27], JARVIS2 [15], GeCo3 [16], UHT [28], NUHT [29], DeepDNA [30], and XM [31]. Reference-based methods often achieve superior compression ratios, particularly for genome sequences from the same species with long read lengths. However, these methods rely on the availability of a high-quality reference genome, which is also required for decompression. If the reference genome is not representative of the sequences being analyzed, it can introduce bias and lead to inaccuracies. In contrast, reference-free methods are self-contained and not only reduce storage requirements but also support advanced genomic and metagenomic analyses. For instance, studies [32, 33, 34] have demonstrated the utility of compression-based features, such as Normalized Compression (NC), for taxonomic classification.

The introduction of the FASTA format standardized genomic data representation, combining sequences with annotations (headers) [15], where nucleotide or amino acid sequences constitute the majority of the data. Various tools, such as MBGC [35], NAF [36], MFCompress [37], and Deliminate [38], employ specialized compression algorithms with simple header coding. These algorithms leverage repeated and inverted regions using techniques like bit encoding, context modeling, dictionary methods, and statistical approaches, including Markov models, run-length encoding, and Huffman coding. Recently, there has been a trend toward developing learning-based genome compression methods that utilize neural networks [25, 26, 27, 30, 39, 40] as well as well as fusion-based approaches [15, 16] that combine specialized DNA models with neural networks. While achieving improved results, they face challenges such as high computational complexity, extended runtimes, limited generalization, interpretability issues, overfitting, and sensitivity to hyperparameters. Furthermore, the integration of multiple models for efficient DNA compression remains a complex problem.

In this paper, we take a different approach by introducing HMG (heuristic-driven MDL-based Genome sequence compressor), a novel genome compressor that integrates tailored heuristic-based algorithms—specifically Genetic Algorithm (GA) [41] and Simulated Annealing (SA)

[42]—with the Minimum Description Length (MDL) principle [43]. The MDL principle posits that "the best model for a given dataset is the one that most effectively compresses the data while minimizing the total length of the description of both the model and the encoded data." For genome sequences, this means identifying an optimal model that efficiently encodes the sequences. HMG employs a structure called a k-mer table (*KT*) to store each model. The *KT* stores all the k-mers (also referred to as patterns) that best represent the data and defines the encoding scheme for these patterns. However, identifying optimal patterns in genome sequences is challenging due to the vast search space and the abundance of redundant patterns, which makes exhaustive exploration computationally infeasible. To overcome these challenges, HMG leverages heuristic approaches using GA and SA for pattern selection. These algorithms, termed as HMG-GA and HMG-SA, effectively guide the search for optimal patterns while discarding less promising ones, resulting in improved compression.

In previous work [44], we introduced GMG (GA for MDL-based genome compression) as a proof-of-concept demonstrating the potential of heuristic search in genome compression. GMG utilized a GA to direct the search toward promising k-mers, contributing to improved bits-per-base compression and computational efficiency without exhaustively evaluating all possible combinations. Moreover, optimal k-mers identified by GMG can be used for genome classification. While effective, GMG had three key limitations: (1) it did not save all k-mers required for decompression, making reconstruction unfeasible, (2) it employed a basic GA implementation with limited exploration capabilities, and (3) the classification approach relied on basic classifiers, which may limit the overall performance. This paper builds upon GMG by proposing HMG, which achieve better compression and support decompression. To better explore the search space and select more efficient k-mers for encoding, HMG offers two improved heuristics search methods: one based on GA and another based on SA. Furthermore, HMG employs a broader range of crossover and mutation operators to explore their impact on the overall performance of GA in MDL-based genome compression. By offering the flexibility to choose between SA and GA, HMG enables a more thorough exploration of the search space, leading to the identification of truly optimal patterns and improved compression outcomes. Finally, a custom classifier is developed using the k-mers identified by HMG-GA and HMG-SA, which consider their distributions in multiple datasets.

The main contributions of this work are:

1. **Heuristic-driven discovery of optional k-mers**: A GA-based algorithm is introduced to find optimal k-mers for compressing genome sequences with four variants based on various crossover (single-point and cycle crossover) and mutation (standard and scramble mutation) operators. Moreover, a second heuristic approach is proposed, utilizing SA for MDL-based compression.
2. **Optimal k-mers substitution and encoding**: For compression, discovered optimal k-mers are first replaced with unique symbols, and both the optimal k-mers and remaining nucleotide bases from parts of the sequence(s) are encoded using Huffman encoding based on their occurrences. The genome sequences are then saved into binary files. For decompression, the binary files are decoded using the Huffman table to recover the substituted sequence, which are then reconstructed using the optimal k-mers and nucleotides mappings to restore the original genome sequences.
3. **Custom Classifier**: In addition to compression, the derived optimal k-mers are utilized for classification. They serve not only as features for traditional classifiers but also support the development of a custom classifier. Training this classifier specifically on the k-mers can

effectively distinguish between various classes. This approach provides a more specialized and potentially more accurate classification model compared to traditional methods.

4. **Experiments**: Extensive experiments are performed on four diverse datasets to evaluate the developed framework for various metrics, including compression in terms of bits-per-base and reduced size, compression and decompression runtime, the number of generations required by GA and SA to achieve compression and classification. Lastly, the obtained results are compared with the state-of-the-art genome compressors.

The rest of the paper is organized as follows: Section 2 presents the literature review, providing an overview of existing methods for genome sequence compression, followed by a brief introduction to the heuristic algorithms (GA and SA). Section 3 discusses how the MDL principle can be applied to genome sequence compression. Section 4 describes the proposed HMG compressor for genomes. Section 5 presents the experimental evaluation of HMG against state-of-the-art genome sequence compressors. Finally, Section 6 provides the conclusion.

2. Literature Review

This section contains two parts. Section 2.1 reviews related work on reference-free genome compression. Then, Section 2.2 presents a brief introduction to GA and SA, to describe the background of this work.

2.1. Reference-free Genome Compressors

Reference-free methods focus on compressing genome sequences independently, aiming to utilize algorithms and heuristics to identify and exploit redundancies without relying on external references. In recent years, efficient methods have been developed that not only improve compression efficiency but also facilitate the analysis of complex genomic variations. This makes reference-free approaches particularly valuable in fields such as personalized medicine, evolutionary biology, and metagenomics. Biocompress [17] is the first reference-free genome compressor that paved the way for the development of specialized compressors specifically designed to address the unique challenges associated with compressing genomic sequences. It utilized Ziv and Lempel algorithms to detect palindromes and repeats in the target genome, which are then encoded based on the length and position of their earliest occurrences. Next, we discuss reference-free genome compressors published in the last six years.

DeepDNA [30] achieved effective compression of human genome data using a hybrid deep learning model. This model combines convolutional layers to capture local features with recurrent layers to model long-term dependencies in genomic sequences. NAF [36] is a reference-free lossless compressor for both FASTA and FASTQ sequences. For Compression, NAF divides the input into headers, sequences, masks, and qualities, which are processed separately. The sequences are combined and converted into 4-bit encoding, which is then compressed using the ZSTD compressor to produce compact data streams. For decompression, NAF reverses this process by decompressing the separate streams and then reassembling them. GeCo3 [16] implemented a fusion of specialized DNA models (the context and substitution-tolerant context models) with a neural network to enhance genomic sequence compression. Alyami et al. [29] improved Huffman encoding by employing a nongreedy unbalanced Huffman tree (NUHT). This approach achieves superior compression ratios and speed compared to traditional methods like GZIP, BZIP2, and UHT [28], which employ an unbalanced Huffman tree for encoding. Cui et al. [40] employed a CNN and an attention-based bidirectional LSTM and arithmetic encoder to

compress genome sequences. Cao et al. proposed the XM [31] algorithm for biological sequence compression. It fuses expert probabilities based on statistical properties and sequence repetitions, encoding symbols with arithmetic coding.

JARVIS2 [15] provided a fusion of finite-context models, stochastic repeat models, and a neural network to achieve efficient genomic data compression with reduced computational resources. Its successor, JARVIS3 [25], enhances compression performance through improved table memory models and probabilistic lookup tables. GraSS [24] incorporated specific characteristics of DNA by leveraging grammatical, statistical, and substitution rules. However, it relies on the general-purpose BSC compressor to perform the final compression on the encoded sequences. The NanoSpring [45] compressors for nanopore read sequences rely on an approximate-assembly approach. The LEC-Codec [26], a learning-based genome codec, utilizes deep neural networks with group of bases compression, multi-stride coding, and bidirectional prediction to achieve efficient and flexible lossless genomic data compression. Similarly, GenCoder [27] was introduced as a deep learning-based algorithm that utilizes a convolutional autoencoder for reference-free lossless compression of genomic sequences. DNACoder [39] employs a CNN-LSTM attention-based prediction network to compress the genomic data. GeneSqueeze [46] employs the inherent patterns found in the fundamental elements of FASTQ files, such as quality scores, nucleotide sequences and read identifiers. It offers numerous advantages, such as an adaptive compression protocol tailored to the specific distribution of each sample, lossless retention of IUPAC nucleotides and read identifiers, and compatibility with all FASTQ/A file attributes. A summary of the discussed genome compressors along with their key parameters is presented in Table 1.

Reference-free methods have emerged as valuable tools for taxonomic classification, offering efficient and scalable approaches to analyze and organize genomic data without relying on predefined reference genomes. These methods are particularly beneficial for metagenomics, where the diversity of unknown organisms presents unique challenges. Taxonomic classification is essential for understanding biodiversity, evolutionary relationships, and the ecological roles of organisms. One such study [32] introduced a novel reference-free approach for metagenomic identification, utilizing features from multiple data compressors for taxonomic classification. Similarly, studies [33, 34] proposed classification approaches for Archaea and viral taxa, respectively. Both studies employed various features, including NC, sequence length, GC-content, and the percentage of each nucleotide. Their findings indicated that combining all these features together yielded better classification results compared to using a single feature or a subset of features. Sukru et al. [47] proposed a compressor-based and parameter-free method for DNA sequence classification. By utilizing algorithms such as GZIP, Snappy, Brotli, LZ4. ZSTD, BZ2 and LZMA, this method achieved high accuracy in species classification while being more resource-efficient than traditional machine learning approaches. A notable aspect of [32, 33, 34, 47] is the utilization of a compression-based feature, NC, which measures the similarity between sequences based on their compressibility. NC provides a quantitative metric for determining the relatedness of different genomic sequences by analyzing how well sequences compress together. This feature is particularly useful for taxonomic classification, as it allows a more accurate assessment of sequence similarity by leveraging the inherent patterns. Consequently, it facilitates the identification of evolutionary relationships and the clustering of similar organisms based on their genomic content.

Reference-free genome compressors have made significant progress in recent years. Nonetheless, the inherent complexity of identifying an optimal set of k-mers presents additional challenges. To address these challenges effectively, heuristic algorithms can be employed.

Table 1: Overview of related works on reference-free genomic compressors

Study	Method	Cla	Strengths	Limitations
Biocompress [17]	Using Ziv and Lempel algorithms to identify palindromes and repeats.	No	First reference-free genome compressor; identifies genomic patterns based on length and position.	As the first DNA compressor, it is not as efficient compared to later methods.
DeepDNA [30]	Hybrid deep learning model combining convolutional and recurrent layers.	No	Effective compression of human genome data; captures both local features and long-term dependencies.	Computationally intensive and requires large datasets.
NAF [36]	Splits input into headers, sequences, masks, and qualities, converts sequences into 4-bit encoding.	No	Separates input into headers, sequences, and qualities for efficient compression; uses 4-bit encoding and ZSTD for final compression.	Depends on external compressor, computationally complex and not suitable for very small datasets.
GeCo3 [16]	Neural networks with specialized DNA models.	No	Enhances genomic sequence compression by mixing multiple contexts and substitution-tolerant models.	Computationally expensive and complex to implement.
NUHT [29]	Nongreedy unbalanced Huffman tree.	No	Superior compression ratios and speed compared to GZIP, BZIP2, and UHT	May not be as effective on all types of genomic data.
Cui et al. [40]	CNN and attention-based bi-directional LSTM combined with arithmetic encoder.	No	Combines deep learning models and encoding techniques for efficient genome sequence compression.	Neural network-based, so computationally expensive and requires large datasets for training.
XM [31]	Expert probabilities and arithmetic coding.	No	Outperforms existing compressors on DNA and protein datasets.	May not be as effective on non-biological data or very large datasets.
JARVIS2 [15]	Finite-context models, stochastic repeat models, neural networks.	No	Efficient compression with reduced computational resources.	May not scale well for large datasets.
JARVIS3 [25]	Extension of JARVIS2.	No	Enhanced models for table memory models as well as lookup-tables in repeat models for better optimization.	May not scale well for large datasets.
NanoSpring [45]	Use an approximate assembly approach.	No	3-6x better compression than gzip; faster decompression with multiple threads.	Limited to base sequences in FASTQ, does not process quality scores or other data.
LEC-Codec [26]	Learning-based genome codec.	No	Utilizes deep neural networks, multi-stride coding, and bidirectional prediction for efficient compression.	May require significant computational resources for training the neural networks.
GenCoder [27]	Deep learning-based convolutional autoencoder.	No	Achieves 27% compression gain over state-of-the-art methods; reconstructs data from latent code.	Limited to genomic data; may not generalize to other biological data.
DNACoder [39]	Deep learning-based CNN and LSTM.	No	Achieves 21.1% better compression than existing compressors	Computationally expensive and requires large datasets for training.
GeneSqueeze [46]	Use nucleotide sequences, quality scores, and read identifiers.	No	Achieves up to 3x better compression than gzip with auto-tuning for sample distribution.	Limited to FASTQ/A files; does not handle other sequencing file formats.

Cla: Classification

2.2. Heuristic Algorithms

Heuristic algorithms can find approximate solutions to complex problems within a reasonable time. These algorithms iteratively refine candidate solutions to optimize a given problem. While they do not guarantee an optimal solution, they provide a feasible solution that is good enough for practical purposes. Due to their efficiency and practicality, heuristic algorithms are not only used in optimization problems but are also widely employed in other areas, including scheduling problems, robotics, bioinformatics, control engineering and artificial intelligence. We next discuss the two heuristic algorithms (GA and SA) that are employed in this work.

2.2.1. Genetic Algorithms

GAs, a class of heuristic search algorithms, are inspired by the principles of genetics and natural selection ('survival of the fittest'). Developed by John Holland [41] in the 1970s, GAs are designed to solve complex problems by mimicking the evolutionary processes that drive biological adaptation. They follow the process of evolution, where the fittest individuals are selected to reproduce and pass their genetic information to the next generation. They work with a population of potential solutions, termed chromosomes, by applying genetic operators such as selection, crossover, and mutation [48, 49] . During selection, chromosomes are chosen for reproduction based on their fitness, while crossover combines genetic material from two parent chromosomes to create new child chromosomes. Mutation, on the other hand, is a random alteration of a solution. By iteratively refining the population, GAs explore a large solution space and converge on high-quality solutions to complex problems. In other words, GA evolves a population of individuals (chromosomes) that are better adapted to their environment than the individuals from which they were derived.

One of the key advantages of GAs is their ability to handle large, complex search spaces without relying on gradient information, making them well-suited for problems that are infeasible to solve using traditional methods. Furthermore, GAs are highly adaptable and can be easily customized to suit the specific requirements of different optimization tasks. Their inherent parallelism allows them to evaluate multiple solutions simultaneously, which enhances their efficiency and robustness. Furthermore, the flexibility of GAs enables the customization of operators and parameters to suit specific problem domains, leading to improved performance in specific contexts.

2.3. Simulated Annealing

SA [42, 50], also a heuristic optimization algorithm, mimics the physical process of annealing in metallurgy, where a metal is heated and then slowly cooled to reduce its defects and achieve a lower energy state. From an optimization perspective, SA involves several key steps: problem configuration, neighborhood exploration, objective function evaluation, and the cooling/annealing process. More specifically, SA starts with an initial solution and iteratively explores the solution space by making small, random changes to the current solution. Changes are accepted or rejected based on a probability that depends on the difference in quality between the new and current solutions, as well as a temperature parameter that gradually decreases over time. This cooling schedule allows SA to initially explore a wide range of solutions and gradually converge on a high-quality solution as the temperature decreases.

The ability of SA to escape local minima and to explore a diverse set of solutions makes it particularly well-suited for problems with complex, non-convex landscapes. Furthermore, SA is relatively easy to implement and can be applied to both continuous and discrete optimization problems. SA is very similar to the Hill-Climbing algorithm [51], with one key difference: at high temperatures, SA allows for the acceptance of worse neighbors, which helps to avoid getting trapped in local optima [52].

3. Adapting the MDL Principle for Genome Sequence Compression

The proposed compressor is grounded in the MDL principle, which provides a framework for selecting models that best represent a genome dataset by balancing model complexity and data fit. In the context of genome compression, MDL helps identify the smallest set of k-mers that

effectively capture the sequential patterns of the genome while minimizing the total compressed size. This section first explains how MDL is adapted for genome compression, followed by discussing the potential of employing GA and SA as heuristic strategies to enhance compression performance.

Definition 1 (MDL)**.** For a set of models $\mathbb{M}$, the best model $M \in \mathbb{M}$ for describing a database $\mathbb{D}$ is the one that minimizes the total compressed size of $\mathbb{D}$ [43], given by:

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

where $L(M)$ is the model's description size (in bits), and $L(\mathbb{D}|M)$ is the compressed size of $\mathbb{D}$ using M.

To employ the MDL principle for genome compression, it is necessary to establish the structure of a model $M \in \mathbb{M}$, its utilization in describing a database, and the encoding of this model in bits. Key definitions are as follows:

Definition 2 (Nucleotide base set)**.** *Let $NB = \{A, C, G, T\}$ represent the set of nucleotide bases.*

Definition 3 (Genome sequence, Database)**.** *A genome sequence S is an ordered list of bases from NB. A database $\mathbb{D}$ is a collection of one or more genome sequences [53]. The i-th base of a genome sequence S is denoted as $S[i]$.*

Definition 4 (k-mer)**.** *A k-mer $kM = (B_1, B_2, \ldots B_k)$ is a sequence of k bases from NB.*

Definition 5 (k-mer occurrence)**.** *An occurrence (occ) of a k-mer $kM = (B_1, B_2, \ldots, B_k)$ in a genome sequence S is a list of consecutive indices $(i_1, i_2, \ldots i_k)$ such that:*

$$S[i_1] = B_1, S[i_2] = B_2, \ldots, S[i_k] = B_k$$

A k-mer can have multiple occurrences in a genome sequence S. Two occurrences are non-overlapping if their index sets do not intersect.

A model M is a set of k-mers used for compressing a genome sequence database $\mathbb{D}$, and is represented by a structure called a *k-mer table*, denoted as KT. The KT is crucial in the proposed approach, as compressing the database is achieved by replacing k-mer occurrences in the database by their codes from the KT.

Definition 6 (k-mer table)**.** *A* k-mer table *KT is a dictionary where each k-mer kM from KT is paired with a unique binary code $code(kM)$. Formally, $KT \subseteq \{(kM, code(kM)) \mid kM \in CS\}$, where $CS = \{kM_1, kM_2, \ldots, kM_m\}$ is the set of k-mers in KT, called coding set.*

The KT only contains k-mers that occur at least once in the database. Because k-mers from the KT may have overlapping occurrences in sequences, the order that k-mers are used for compressing a database influences the compressed size. To achieve optimal compression, the KT is ordered first by descending usage (defined next), then by descending k-mer length, and finally by ascending lexicographical order (for breaking ties). This order called *k-Mer Cover Order* is a variation of the *Standard Cover Order* [54]. The matching of k-mers from KT to a genome sequence for compression is done through a cover function:

Definition 7 (Cover function)**.** *Let $\mathbb{D}$ be a database of sequences over NB, and KT be the k-mer table. The* cover function*, denoted as $cover(KT, S)$, maps each sequence $S \in \mathbb{D}$ to a set of non-overlapping k-mers from KT that fully reconstruct S when concatenated. The cover function must satisfy two conditions:*

1. ***Membership**: All k-mers in the cover belong to KT:*

$$kM \in cover(KT, S) \Rightarrow kM \in KT$$

2. ***Completeness**: The concatenation of k-mers in the cover can entirely reconstruct S, ensuring complete coverage:*

$$S = \bigodot_{kM \in cover(KT,S)} kM$$

where $\odot$ denotes the concatenation of k-mers. This concatenation process is performed greedily, such that k-mers are concatenated in the k-mer Cover Order to reconstruct S, and where k-mers are not allowed to overlap.

Definition 8 (Usage)**.** *The usage count of a k-mer $kM \in KT$ for a database $\mathbb{D}$ is:*

$$usage(kM) = \sum_{S \in \mathbb{D}} count_{kM}(S) \tag{2}$$

where $count_{kM}(s)$ is the usage count of kM in S, defined as the number of occurrences of kM that are utilized for reconstructing the sequence S using the cover function.

To clarify the concept of the KT, cover function and their usage for compression, a small example is provided in Figure 1. In this example, the genome sequence $TTGTACGTTTTGT$ is encoded using a KT containing three k-mers (TT, GT and ACG) and their codes. Codes are depicted as colored bars, where bar widths represent the relative lengths of codes. The encoded sequence is obtained by replacing k-mers in the original sequence by their codes. This encoding is done based on the cover function, which is represented by the arrows, and defines the mapping of k-mers to the genome sequence. This mapping is such that k-mers do not overlap and cover the whole sequence.

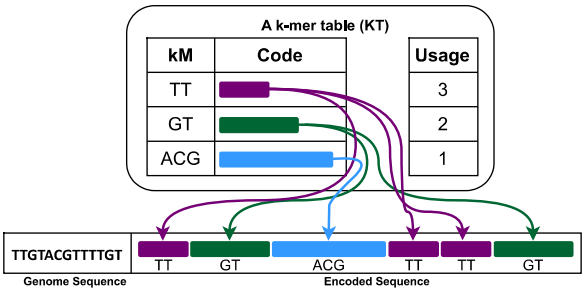


Figure 1: An example of a genome sequence (bottom left) and its corresponding encoding (bottom right) using the k-mer table (KT) (top). The KT contains k-mers with their respective codes visually represented as bars, where the widths of the bars indicate the lengths of the codes. The arrows represent the mapping provided by the cover function.

To optimally compress a database $\mathbb{D}$ using the MDL principle (eq. 1), it is necessary to take into account two components: the encoded size of the database $\mathbb{D}$ and the size of the model KT. The size of the model KT is determined by summing the code lengths of all k-mers in KT that have a non-zero usage:

$$L(KT|\mathbb{D}) = \sum_{\substack{kM \in KT \\ usage(kM) \neq 0}} L(kM|KT) \tag{3}$$

Frequent k-mers receive shorter codes. The optimal code length $L(kM|KT)$ for a k-mer kM can be derived from its empirical probability in $\mathbb{D}$ based on the Shannon entropy as:

$$L(kM|\mathbb{D}) = -log_2(P(kM|\mathbb{D})) \tag{4}$$

where the probability $P(kM|\mathbb{D})$ reflects the relative usage of kM for encoding $\mathbb{D}$:

$$P(kM|\mathbb{D}) = \frac{usage(kM)}{\sum_{kM \in KT} \text{usage}(kM)} \tag{5}$$

For any sequence S, its encoded size $L(S|KT)$ is calculated as the sum of the code lengths of all k-mers kM in its cover multiplied by their respective usage count in S:

$$L(S|KT) = \sum_{kM \in cover(S)} L(kM|KT) \cdot \text{count}_{kM}(S) \tag{6}$$

The encoded size of the entire database $\mathbb{D}$ given KT is then the sum of the encoded sizes of all sequences in $\mathbb{D}$:

$$L(\mathbb{D}|KT) = \sum_{S \in \mathbb{D}} L(S|KT) \tag{7}$$

The optimal set of k-mers are the one whose associated KT minimizes the total compressed size, defined as:

Definition 9 (Total compressed size)**.** *The total compressed size of a database ($\mathbb{D}$) and k-mer table (KT) in bits is:*

$$L(\mathbb{D}, KT) = L(\mathbb{D}|KT) + L(KT|\mathbb{D}) \tag{8}$$

To effectively compress genome sequences within a database $\mathbb{D}$, we aim to identify the k-mer set that best describe $\mathbb{D}$. This objective is formalized using $L(\mathbb{D}, KT)$ as follows:

Minimal Ordered k-mer Set Problem: Let NB be the set of nucleotide bases and $\mathbb{D}$ be a database of sequences over NB. Let $\mathcal{KM}$ denote the collection of all possible k-mers that can be derived from NB. The minimal ordered k-mer set problem is to identify the smallest subset of k-mers (or patterns) $P \subseteq \mathcal{KM}$ that minimizes the total compressed size $L(\mathbb{D}, KT)$, where KT is the k-mer table containing P.

Utilizing MDL for genome compression necessitates the identification of an optimal set of k-mers and their corresponding codes to construct a KT. However, the combinatorial explosion of potential k-mer sets makes exhaustive search infeasible. To address this, HMG employs two heuristic strategies, prioritizing models with lower $L(\mathbb{D}, KT)$: **GA**: Evolves candidate k-mers through crossover and mutation, **SA**: Iteratively refines candidate k-mers by exploring their neighbors using a temperature-based mechanism to escape local minima. Both heuristics aim to minimize $L(\mathbb{D}, KT)$, ensuring a direct trade-off between compression efficiency $L(\mathbb{D}|KT)$ and model complexity $L(KT|\mathbb{D})$.

Practical Relaxation of Coverage: While the described theoretical framework assumes that the k-mer table KT can fully cover all sequences in $\mathbb{D}$, heuristic searches may fail to find such a complete set. To handle this, the proposed algorithms allows parts of sequences to remain uncovered by KT. These uncovered regions are not compressed (remain as raw nucleotide bases from NB) alongside the encoded k-mers. This relaxation ensures compression remains feasible even when no perfect k-mer set is found, though it increases $L(\mathbb{D}|KT)$. The MDL principle still guides optimization by explicitly penalizing both residual uncompressed data (through $L(\mathbb{D}|KT)$) and excessive model complexity (through $L(KT|\mathbb{D})$).

4. The HMG compressor

This section introduces the proposed Heuristic-driven MDL-based Genome sequence compressor (HMG), designed to discover a representative set of k-mers that effectively compress a set of genome sequence(s), and also support both compression and decompression. The schematic of HMG is shown in Figure 2. The framework is divided into four main phases: (1) initial k-mers generation and evolution, (2) compression, (3) decompression, and (4) classification. The main operations performed in each phase are as follows:

1. **Initial k-mers Generation and Evolution (highlighted in Yellow)**: In the first phase, genome sequence(s) are received as input. Then, an initial random set of k-mers is created from the four bases A, C, G, and T, for compressing the genome sequence(s). These k-mers are the starting point to search for a more optimal set of k-mers using heuristics. HMG employs two heuristic algorithms, GA and SA, to iteratively evolve the k-mers. The fitness of each generated k-mer is evaluated, with priority given to those that occur more frequently in the genome(s). High-occurrence k-mers are added to the KT, contributing significantly to compression. Once the stopping criteria are met, the final KT is passed to the next phase.
2. **Compression (highlighted in Red)**: In this phase, the representative optimal k-mers in the KT along with the nucleotide bases from parts of the sequence(s) not covered by KT are encoded using Huffman encoding to generate a binary output file. This binary file, along with a dictionary that maps the original sequence(s) to compressed patterns, forms the compressed representation of the genome(s).
3. **Decompression (highlighted in Blue)**: During decompression, the binary file is decoded using the dictionary to accurately reconstruct the original sequence(s). This phase ensures lossless compression.
4. **Classification (highlighted in Green)**: In this phase, the derived k-mers in the KT are utilized for classification tasks. These patterns demonstrate their utility in distinguishing between different genome(s) datasets or identifying specific characteristics of the genome.

Section 4.1 and 4.2 explain in details how the HMG-GA and HMG-SA algorithms implement the phase 1 of HMG. Then, the encoding scheme used for compression (phase 2) and decompression (phase 3) is described in Section 4.3. The phase 4 (classification) is presented in Section 5.3.

4.1. HMG-GA

HMG-GA starts by creating initial k-mers from bases, which serve as potential solutions for compressing the dataset $\mathbb{D}$. This initial population of k-mers is then refined through an iterative GA process that comprises the following steps:

1. **Selection**: Randomly chosen k-mers, referred to as parents, are selected from the initial k-mers to undergo crossover and mutation operations.
2. **Crossover**: The selected parents undergo crossover, where sub-sequences from the parents are combined to create new k-mers, known as children. This process mimics genetic recombination and helps explore new areas of the solution space.
3. **Mutation**: The child k-mers obtained after crossover undergo random modifications. These changes increase the diversity within the population.

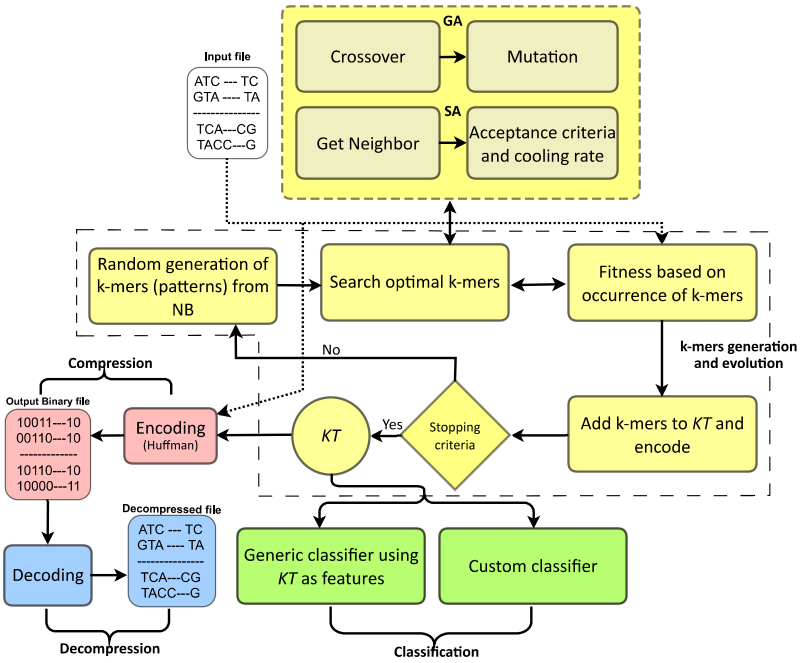


Figure 2: HMG for genome sequence compression contains four main phases: (1) Constructing a *KT*, that contains optimal k-mers discovered by combining heuristic algorithms with the MDL principle, k-mers substitution and encoding (Yellow), (2) Compression (Red), (3) Decompression (Blue), and (4) Using the derived optimal k-mers for genome classification (Green)

4. **Evaluation**: The fitness of the k-mers obtained after crossover and mutation is evaluated by measuring their occurrence frequency in $\mathbb{D}$. The k-mers with higher occurrence frequencies are assigned shorter codes, leading to better compression.
5. **Stopping Criteria**: HMG-GA terminates when a predefined condition is met, specifically when adding k-mers to the KT achieves a satisfactory compression ratio.

Algorithm 1 provides the pseudo-code for HMG-GA, which takes two inputs: (1) a genome sequence dataset $\mathbb{D}$ and (2) a threshold parameter, *maxKTSize*, that defines the maximum size of the KT. The termination condition is that HMG-GA stops once the KT reaches the specified size, effectively limiting the search space and reducing computational time. Additionally, the *maxKTSize* parameter offers users the flexibility to control the number of output k-mers.

HMG-GA outputs a KT containing representative k-mers that efficiently compress the database $\mathbb{D}$. Alternatively, HMG-GA can be configured to execute without the *maxKTSize*, making it fully parameter-free. In this case, HMG-GA terminates when the compression ratio reaches a stable point, particularly when the ratio does not improve over a predefined number of generations.

First, a population is initialized with four bases (line 1) and KT is initialized as empty (line 2). A `while` loop runs till the KT's size reaches the $maxKTSize$ threshold. Two random k-mers P_1 and P_2 with k lengths between 2 and 6 are created in each iteration (line 4). Lines 5-6 count the occurrences of generated k-mers. The k-mers are evolved by applying first the crossover (line 8) and then the mutation (lines 10, 17) operator. Before applying the mutation operator, a random probability check (line 9) determines which offspring undergoes mutation. If the probability is

Algorithm 1 HMG-GA

Input: $\mathbb{D}$: Genome sequence dataset, *maxKTSize*: Maximum k-mer table size
Output: KT: A k-mer table

1: $NB \leftarrow \{A, C, G, T\}$
2: $KT \leftarrow \emptyset$
3: **while** len(KT) $\neq maxKTSize$ **do**
4: $P_1, P_2 \leftarrow$ Two random unique k-mers (length [2,6])
5: $Pocc_1 \leftarrow$ CountOccurrences($P_1, \mathbb{D}$)
6: $Pocc_2 \leftarrow$ CountOccurrences($P_2, \mathbb{D}$)
7: **repeat**
8: $C_1, C_2 \leftarrow$ Crossover(P_1, P_2)
9: **if** *Random*() > 0.5 **then**
10: $C_1 \leftarrow$ Mutation(C_1)
11: $Cocc_1 \leftarrow$ CountOccurrences($C_1, \mathbb{D}$)
12: **if** $Cocc_1 > Pocc_1$ **then**
13: $P_1 \leftarrow C_1$
14: $Pocc_1 \leftarrow Cocc_1$
15: **end if**
16: **else**
17: $C_2 \leftarrow$ Mutation(C_2)
18: $Cocc_2 \leftarrow$ CountOccurrences($C_2, \mathbb{D}$)
19: **if** $Cocc_2 > Pocc_2$ **then**
20: $P_2 \leftarrow C_2$
21: $Pocc_2 \leftarrow Cocc_2$
22: **end if**
23: **end if**
24: **until** (maximum number of generations is reached)
25: $KT \leftarrow KT \cup \{P_1, P_2\}$
26: Replace P_1, P_2 in $\mathbb{D}$ with codes
27: **end while**
28: **Return** KT

greater than 0.5, C_1 is selected for mutation, and its number of occurrences ($Cocc_1$) is counted (line 11). Otherwise, C_2 is mutated, and its number of occurrences ($Cooc_2$) is counted (line 18). The occurrence frequencies (either $Cocc_1$ or $Cocc_2$) are compared with those of their parent k-mers. The parent k-mers are replaced by the evolved k-mers if the latter occur more (lines 14, 20). The evolution process continues through multiple generations, gradually improving the results. After a desired number of generations, the evolved k-mers are added to the KT (line 25). In the KT, unique codes, based on occurrence frequencies, are assigned to the k-mers, which are then utilized for encoding the database $\mathbb{D}$ (line 26). The final output consists of the KT, a collection of k-mers that efficiently compress the dataset.

Counting the occurrences of C_1 and C_2 can be particularly time-consuming, especially with large datasets. To enhance efficiency, we incorporated a probabilistic approach (line 12). Instead of calculating the occurrences for both k-mers, a 50% probability is assigned to either C_1 or C_2 being selected, thereby evaluating only one k-mer per iteration. This approach significantly decrease the computational cost.

A notable characteristic of HMG-GA is its capacity to maintain or enhance solution quality across iterations. As detailed in Algorithm 1 (lines 12-15 and 19-22), the primary optimization metric, the compression ratio, either improves or remains stable. This is accomplished through a selective update process: parent k-mers are replaced with the child (evolved) k-mers if the latter provides a better compression ratio. Otherwise, the parent remains unchanged. This makes sure that the quality of the discovered optimal k-mers does not reduce over time. Further discussion of the process is provided in the results section (see convergence results in section 5.2).

Algorithm 2 Single-Point Crossover

Input: PkM_1, PkM_2: Two parent k-mers
Output: CkM_1, CkM_2: Two child k-mers

1: **procedure** SPC(PkM_1, PkM_2)
2: $s \leftarrow \min(\text{len}(PkM_1), \text{len}(PkM_2))$
3: $cp \leftarrow \text{randomInt}(1, s)$ // $1 \leq cp \leq s$
4: $CkM_1 \leftarrow PkM_1[1, cp] \odot PkM_2[cp + 1, \text{len}(PkM_2)]$
5: $CkM_2 \leftarrow PkM_2[1, cp] \odot PkM_1[cp + 1, \text{len}(PkM_1)]$
6: **Return** CkM_1 and CkM_2
7: **end procedure**

In HMG, the crossover operation combines two parent k-mers to generate child k-mers by exchanging segments of their sequences. The two crossover operators utilized within HMG-GA are presented in Algorithm 2 and 3, and are explained with simple examples next. Let PkM_1 and PkM_2 be two k-mers:

$$PkM_1 = GTCATC$$
$$PkM_2 = CGATG$$

For Single-Point Crossover (SPC) (Algorithm 2), one crossover point (cp) is randomly selected and the genes to the left (or right) of cp are swapped to obtain two new child genes CkM_1 and CkM_2. Let n represent the length of the smallest k-mer. A random cp ($1 \leq cp \leq n$) is randomly selected. For $cp = 3$, applying SPC on PkM_1 and PkM_2 generates the two child k-mers CkM_1 and CkM_2 as:

$$CkM_1 = GTCTG$$
$$CkM_2 = CGAATC$$

For the Cycle Crossover (CC) operator (Algorithm 3), the process begins by detecting cycles between the two parent k-mers. The cycle detection starts at the first position (index 0) of PkM_1. The value at this position is recorded from PkM_2, and the next position to visit is identified by finding this value in PkM_1. This process continues, moving to the corresponding positions in both parent k-mers until a cycle is formed, meaning that the process has returned to a previously visited position.

In the example with PkM_1 and PkM_2, the cycle detection starts with index 0 of PkM_1, where the value is 'G'. In PkM_2, the value at index 0 is 'C'. 'C' is found at index 2 of PkM_1, so the current index becomes 2. The value at index 2 of PkM_2 is 'A'. 'A' is found at index 3 of PkM_1, so the current index becomes 3. The value at index 3 of PkM_2 is 'T'. 'T' is found at index 1 of

Algorithm 3 Cycle Crossover

Input: PkM_1, PkM_2: Two parent k-mers
Output: CkM_1, CkM_2: Two child k-mers

procedure CC(PkM_1, PkM_2)
  $s \leftarrow \min(\text{len}(PkM_1), \text{len}(PkM_2))$
  $visited, CKM_1, CKM_2 \leftarrow \emptyset$
  $i \leftarrow 0$
while $i < s$ **do**
    if i is not in $visited$ **then**
      $current\_index \leftarrow i$
    while $current\_index$ not in $visited$ **do**
      $visited \leftarrow visited \cup \{current\_index\}$
      $CkM_1[current\_index] \leftarrow PkM_1[current\_index]$
      $CkM_2[current\_index] \leftarrow PkM_2[current\_index]$
      $current\_index \leftarrow$ index of $PkM_2[current\_index]$ in PkM_1
    end while
    else // Skip already visited positions
    $i \leftarrow i + 1$
end while
  for each position p in PkM_1 **do**
    if p is not in $visited$ **then**
      $CkM_1[p] \leftarrow PkM_2[p]$
      $CkM_2[p] \leftarrow PkM_1[p]$
  Return CkM_1 and CkM_2
end procedure

PkM_1, so the current index becomes 1. The value at index 1 of PkM_2 is 'G'. Then, 'G' is found at index 0 of PkM_1, completing the cycle as the process returns to index 0. The final child k-mers are:

$$CkM_1 = GTCAG$$
$$CkM_2 = CGATTC$$

The mutation operation [55] then follows the crossover operation and is applied independently to k-mers. It introduces diversity into the population by randomly altering k-mers to explore new areas of the search space. This operation is applied probabilistically, selecting either CkM_1 or CkM_2 with a 50/50 probability, as described in Algorithm 1. This introduces randomness into the search process, helping to avoid premature convergence to local optima. Two mutation operators are considered.

The first mutation operator, referred to as Standard Mutation (SM) in Algorithm 4, modifies a selected nucleotide base within a k-mer by replacing it with a randomly chosen base from NB. For example, in the case of CkM_1 and CkM_2 obtained after SPC, the mutation occurs at index 1 of CkM_1 (mutating the value from 'T' to 'A') and index 3 of CkM_2 (mutating the value from 'A' to 'C'). As a result, the mutated k-mers are:

$$CkM'_1 = GACTG$$
$$CkM'_2 = CGACTC$$

The second mutation operator is Scramble Mutation (ScM) and is presented in Algorithm 5. In this operator, a random subsequence of NBs within the k-mer is selected and scrambled. First, a subsequence size is chosen randomly, and then a random start position is selected in the k-mer. The selected subsequence of bases is scrambled (randomly shuffled) using the "scramble()" function, and the altered subsequence is placed back into the k-mer. For example, for the sequence 'GATCGT', a randomly selected subsequence "ATC" could be scrambled to 'TAC', resulting in a mutated k-mer:

$$CkM'_1 = GTACGT$$

Algorithm 4 Standard Mutation/Get Neighbor

Input: kM: A k-mer
Output: A mutated k-mer

1: **procedure** SM(kM)
2: $i \leftarrow$ randomInt(1, len(kM))
3: $alter \leftarrow$ randomSelect(NB, 1) // 1-mer
4: $kM[i] \leftarrow alter$ // $kM[i] \neq alter$
5: **Return** kM
6: **end procedure**

Algorithm 5 Scramble Mutation

Input: kM: A k-mer
Output: A mutated k-mer

1: **procedure** ScM(kM)
2: $subsetSize \leftarrow$ randomInt(2, len(kM)) // Size of the subset to scramble
3: $startPos \leftarrow$ randomInt(1, len(kM) - $subsetSize$ + 1)
4: $subset \leftarrow kM[startPos, startPos + subsetSize - 1]$
5: $scrambledSubset \leftarrow$ scramble($subset$)
6: $kM[startPos, startPos + subsetSize - 1] \leftarrow scrambledSubset$
7: **Return** kM
8: **end procedure**

These operators are designed to optimize the selection of k-mers for compression by balancing exploration (via mutation) and exploitation (via crossover). The fitness function guides the process by favoring k-mers that occur frequently and contribute significantly to reducing the compressed size. HMG-GA terminates when the stopping condition is met, specifically when the number of k-mers in the KT equals the $maxKTSize$ threshold. A user can adjust this threshold based on his/her desired output size.

4.2. HMG-SA

HMG-SA involves the following steps:

1. **Initial Solution**: A random k-mer is selected as the starting solution for compression.

2. **Neighbor Generation**: A neighboring k-mer is generated by making a small change to the selected k-mer. This can be achieved by altering some part(s) of the k-mer.
3. **Evaluation**: The fitness of the newly generated neighbor k-mer is assessed by measuring its occurrence frequency in the dataset. The goal is to find k-mers with high occurrence frequencies, as these will lead to better compression.
4. **Acceptance Criterion**: If the new k-mer has a better fitness value (higher occurrence frequency), it replaces the parent k-mer. If not, the algorithm calculates an acceptance probability using the temperature parameter. The probability of accepting a child k-mer, that does not improve compression, decreases as the temperature lowers. This probabilistic acceptance allows the algorithm to escape local minima and explore different parts of the search space of k-mers.
5. **Cooling Rate**: The temperature is gradually decreased by a cooling rate parameter, reducing the probability of accepting child k-mers, that do not improve compression, over time. This process helps the algorithm converge to a globally optimal solution as the temperature approaches zero.
6. **Stopping Criteria**: HMG-SA terminates when a predefined condition is met, that is adding k-mers in the *KT* till achieving a satisfactory compression ratio.

Algorithm 6 presents the proposed pseudocode of HMG-SA that is used to find the k-mers to effectively compress the genome sequence(s). Similar to HMG-GA, it starts by initializing the population with the four nucleotide bases (line 1). The *KT* is initialized as an empty set (line 2) and the temperature (T) is set to an initial temperature (T_0) (line 3). A `while` loop runs until the size of *KT* reaches the specified *maxKTSize* threshold. In each iteration, a random k-mer called parent k-mer, *PkM* (with a length between 2 and 6) is generated (line 5), and its occurrences in $\mathbb{D}$ are counted (line 6). The algorithm then enters a nested `while` loop where it iteratively applies the *GetNeighbor* method (line 8) to create a neighboring k-mer called new k-mer (*NkM*). Algorithm 4 outlines the procedure for generating a neighboring k-mer. This function modifies a given k-mer by replacing one of its bases with a randomly selected base from the population (*A*, *C*, *G*, *T*). Specifically, a random index is chosen within the k-mer, and the base at that position is replaced with a new randomly selected base that is different from the original. This operation ensures that small perturbations are introduced into the k-mer, allowing the algorithm to escape local optima and explore alternative k-mer(s). These small changes systematically navigate the search space of k-mers to find potential improvements in the compression model.

The occurrences of the *NkM* are then counted (line 9). If the *NkM* has a higher occurrence count than *PkM* (line 10), *NkM* replaces the *PkM* (line 11), and the occurrence count is updated (line 12). If *NkM* does not improve upon *PkM* (i.e., $Nocc \leq Pocc$), a probabilistic acceptance criterion is applied (line 14). The acceptance probability *acPr* is computed based on the difference in occurrences and the current temperature (which simulates the cooling process). If a random number is less than *acPr*, the *NkM* is accepted, allowing the search to escape local minima (lines 15-17). The initial temperature (T_0) is set to 100 and is gradually reduced by multiplying it by a cooling rate of 0.95. The cooling rate controls the pace at which the temperature decreases, and it plays a crucial role in balancing exploration (searching a wider space) and exploitation (focusing on better k-mer(s)). As the temperature decreases, the algorithm becomes more selective, refining the k-mer(s) to achieve optimal compression.

The inner `while` loop continues until the temperature (T) drops below a threshold (usually set to 1), signaling the end of the annealing process for that particular k-mer. Once T reaches the threshold, the solution *PkM* is added to the *KT* (line 22), and the k-mer is replaced by a code

Algorithm 6 HMG-SA

Input: $\mathbb{D}$: Genome dataset, T_0: Initial temperature, r: Cooling rate, $maxKTSize$: Maximum k-mer table size
Output: KT: A k-mer table

1: $NB \leftarrow \{A, C, G, T\}$
2: $KT \leftarrow \emptyset$
3: $T \leftarrow T_0$
4: **while** len(KT) $\neq maxKTSize$ **do**
5: $PkM \leftarrow$ Random unique k-mer (length [2,6])
6: $Pocc \leftarrow$ CountOccurrences($PkM, \mathbb{D}$)
7: **while** $temperature > 1$ **do**
8: $NkM \leftarrow$ GetNeighbor(PkM)
9: $Nocc \leftarrow$ CountOccurrences($NkM, \mathbb{D}$)
10: **if** $Nocc > Pocc$ **then**
11: $PkM \leftarrow NkM$
12: $Pocc \leftarrow Nocc$
13: **else**
14: $acPr \leftarrow exp((Nocc - Pocc)/T)$
15: **if** $Random() < acPr$ **then**
16: $PkM \leftarrow NkM$
17: $Pocc \leftarrow Nocc$
18: **end if**
19: **end if**
20: $T \leftarrow T \times r$
21: **end while**
22: $KT \leftarrow KT \cup \{PkM\}$
23: Replace PkM in $\mathbb{D}$ with codes
24: **end while**
25: **Return** KT

in $\mathbb{D}$ (line 23). This process continues until the KT reaches the desired size. Finally, the KT, containing the optimal k-mers that minimize the dataset size, is returned (line 25).

The process that distinguishes SA from GA is the annealing process. Unlike GA, which focuses on crossover and mutation, SA uses a probabilistic approach to decide whether to accept a child k-mer. This decision is governed by the acceptance probability, which allows the algorithm to occasionally accept child k-mer(s) that do not perform well compared to the parent k-mer(s). This mechanism enables SA to avoid local optima by exploring potentially suboptimal k-mer(s) that might lead to better global k-mer(s) in the long run. The acceptance probability is determined using the formula:

$$acPr = \exp\left(\frac{Nocc - Pocc}{T}\right)$$

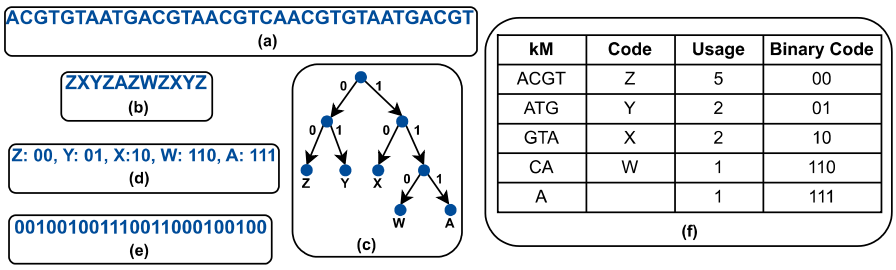
where $Nocc$ is the occurrences of the new (child) k-mer, $Pocc$ is the occurrences of the current (parent) k-mer, and T is the current temperature. The temperature decreases over time according to the cooling rate, reducing the likelihood of accepting child k-mers that do not improve compression as the algorithm progresses.

4.3. Encoding

The encoding method used in the genome compression process is based on substitution encoding, followed by Huffman encoding [28] which is a well-known lossless data compression technique.

First, a substitution encoding is used, where the derived optimal k-mers in the genome(s) dataset ($\mathbb{D}$) are replaced with unique characters. Next, a Huffman tree is constructed based on the frequency of k-mers in $\mathbb{D}$, assigning shorter codes to more frequent k-mers and longer codes to those that are less frequent. Once the Huffman tree is built, a unique binary code is generated for each k-mer which is then used to replace the k-mers in the dataset to produce the encoded output. The Huffman tree ensures that each k-mer is represented by a distinct binary string, and the encoding is prefix-free, meaning no code is a prefix of another. This property is crucial for efficient decoding, as it prevents ambiguity during the decompression process. This fusion of substitution encoding with Huffman encoding enhances the overall compression efficiency by ensuring that the most common patterns are represented in the most compact form possible.

As an example, Figure 3 illustrates the encoding process for a short sequence (a). The process begins by replacing subsequences with codes of k-mers from the *KT* (b). Next, a Huffman tree is constructed (c) to assign shorter binary codes to the most frequently occurring k-mers (d). Finally, the sequence is encoded by substituting the k-mers and remaining nucleotide bases with their corresponding binary codes (e).



kM	Code	Usage	Binary Code
ACGT	Z	5	00
ATG	Y	2	01
GTA	X	2	10
CA	W	1	110
A		1	111

(f)

Figure 3: An example of a genome sequence encoding: (a) A genome sequence, (b) kM substitution, (c) Huffman tree, (d) Binary code, (e) final encoding and (f) kM substitution and Huffman code assignment

In summary, heuristic algorithms (GA and SA), are utilized in HMG to generate candidate k-mers. The optimal k-mers identified during this process are assigned unique characters (not part of the bases). A frequency map of the transformed k-mers is then created, and a Huffman tree is constructed based on the k-mer frequencies. During compression, the transformed k-mers are encoded using these Huffman codes, and the compressed data is written to a binary file. Additionally, a dictionary file is generated, storing the mapping of the original k-mers to their unique characters and the Huffman codes, ensuring efficient decoding during decompression. The final output includes the size of the compressed files, demonstrating the efficiency of the approach while preserving the ability to accurately reconstruct the original genome(s).

5. Experimental Evaluation

In this section, the performance of HMG-GA and HMG-SA is evaluated on 4 datasets and also benchmarked against four reference-free genome compressors: JARVIS2 [15], GeCo3 [16],

JARVIS3 [25] NUHT [29] and the BZIP2 general purpose compressor. 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 32 GB of RAM and 1 TB of disk storage, an adequate configuration for handling the large datasets involved in the experiments. For evaluation and benchmarking, four datasets are used, which are:

- **Dataset 1 (DS1)**: A comprehensive balanced dataset [56] 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) [57], 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.
- **Dataset 3 (DS3)**: This dataset contains the first bacterial genome of the Escherichia coli K-12 substr.MG1655 that was generated by using the MinION sequencer [58]. The whole genome consists 10,132,567 bases.
- **Dataset 4 (DS4)**: This dataset contains extensively sequenced multidrug-resistant isolates of Pseudomonas aeruginosa, Citrobacter freundii and Citrobacter werkmanii from a hospital outbreak [59], all of which contain the carbapenemase enzyme bla$_{IMP-8}$ gene. This dataset contains 1,798,284,785 bases in total.

5.1. Benchmarking Results

This section presents the results in terms of bits-per-base (BPB), compression ratio (CR), compression time, and decompression time across the four datasets. BPB is a measure of the number of bits required to represent each base in the dataset after compression. CR is defined as the ratio of the original file size to the compressed file size:

$$\text{CR} = \frac{\text{Original File Size}}{\text{Compressed File Size}}$$

A higher CR value indicates more efficient compression, as it reflects greater file size reduction (more data is stored in fewer bits). For example, if a file originally takes 100 MB and, after compression, it is reduced to 20 MB, the CR would be 5, meaning that the file has been compressed to one-fifth of its original size. The convergence behavior of HMG-GA and HMG-SA is then

analyzed to observe how it systematically refines the compression ratio over successive iterations for optimal k-mer selection. Finally, the classification results are discussed where first the quality of the optimal k-mers in the *KT* is evaluated by utilizing them as features for classification, followed by the results obtained with the custom classifier developed.

5.1.1. Compression: BPB and CR

The results for compression in terms of BPB and CR obtained with four variants of HMG-GA and HMG-SA are presented in Table 2. In the table, C&M is used to denote the specific combination of crossover and mutation variants employed in the compression process. The same notation is also used in Tables 3 and 4. Overall on average across all datasets, HMG-SA achieves a lower BPB value on average than the four variants of HMG-GA. HMG-SA provided 3.24%, 2.59%, 2.59% and 1.94% better compression than HMG-GA(SPC/SM), HMG-GA(CC/SM), HMG-GA(SPC/ScM) and HMG-GA(CC/ScM) respectively. The difference between the average BPB obtained with the four variants of HMG-GA is very small. Whereas, HMG-GA(CC/SM) variant achieves better CR than other three variants of HMG-GA and HMG-SA. HMG-GA(CC/SM) provided 1.96%, 2.21%, 0.97%, and 5.5% better compression than HMG-GA(SPC/SM), HMG-GA(SPC/ScM), HMG-GA(CC/ScM) and HMG-SA respectively. The difference between the average CR obtained with the four variants of HMG-GA is very small. For benchmarking and comparison with previous compressors, HMG-GA(CC/SM) is selected as it performed slightly better than others for both metric of compression.

Table 2: HMG results for compression in terms of BPB and CR on four datasets

Dataset		**C&M**	**BPB**	**CR**	**C&M**	**BPB**	**CR**
DS1	HoSa	$\frac{SPC/SM}{CC/SM}(SA)$	$\frac{1.58}{1.58}$(**1.53**)	$\frac{3.99}{\mathbf{4.04}}$(4.01)	$\frac{SPC/ScM}{CC/ScM}$	$\frac{1.56}{1.55}$	$\frac{3.97}{4.01}$
	GaGa	$\frac{SPC/SM}{CC/SM}(SA)$	$\frac{1.56}{1.57}$(**1.51**)	$\frac{3.84}{3.93}$(**4.19**)	$\frac{SPC/ScM}{CC/ScM}$	$\frac{1.58}{1.61}$	$\frac{3.90}{3.89}$
	DaRe	$\frac{SPC/SM}{CC/SM}(SA)$	$\frac{1.48}{1.55}$(1.49)	$\frac{4.27}{4.47}$(4.33)	$\frac{SPC/ScM}{CC/ScM}$	$\frac{1.56}{\mathbf{1.41}}$	$\frac{4.21}{\mathbf{4.48}}$
	OrSa	$\frac{SPC/SM}{CC/SM}(SA)$	$\frac{1.66}{1.63}$(**1.57**)	$\frac{4.12}{\mathbf{4.30}}$(4.12)	$\frac{SPC/ScM}{CC/ScM}$	$\frac{1.63}{1.61}$	$\frac{\mathbf{4.03}}{4.21}$
	DrMe	$\frac{SPC/SM}{CC/SM}(SA)$	$\frac{1.58}{1.61}$(**1.49**)	$\frac{4.11}{3.98}$(3.99)	$\frac{SPC/ScM}{CC/ScM}$	$\frac{1.60}{1.55}$	$\frac{\mathbf{4.20}}{3.98}$
	EnIn	$\frac{SPC/SM}{CC/SM}(SA)$	$\frac{1.55}{1.53}$(**1.48**)	$\frac{3.99}{4.14}$(3.91)	$\frac{SPC/ScM}{CC/ScM}$	$\frac{1.58}{1.58}$	$\frac{\mathbf{4.18}}{4.13}$
	ScPo	$\frac{SPC/SM}{CC/SM}(SA)$	$\frac{1.57}{\mathbf{1.52}}$(1.76)	$\frac{\mathbf{3.88}}{3.80}$(3.85)	$\frac{SPC/ScM}{CC/ScM}$	$\frac{1.57}{1.54}$	$\frac{3.81}{3.81}$
	PlFa	$\frac{SPC/SM}{CC/SM}(SA)$	$\frac{1.61}{1.59}$(1.65)	$\frac{4.01}{\mathbf{4.22}}$(4.03)	$\frac{SPC/ScM}{CC/ScM}$	$\frac{1.63}{\mathbf{1.55}}$	$\frac{4.08}{4.21}$
	EsCo	$\frac{SPC/SM}{CC/SM}(SA)$	$\frac{1.73}{1.75}$(**1.50**)	$\frac{3.94}{\mathbf{3.99}}$(3.65)	$\frac{SPC/ScM}{CC/ScM}$	$\frac{1.65}{1.66}$	$\frac{3.89}{3.97}$
	HaHi	$\frac{SPC/SM}{CC/SM}(SA)$	$\frac{1.56}{\mathbf{1.52}}$(1.57)	$\frac{4.02}{4.01}$(3.73)	$\frac{SPC/ScM}{CC/ScM}$	$\frac{1.57}{\mathbf{1.52}}$	$\frac{3.99}{\mathbf{4.04}}$
	AeCa	$\frac{SPC/SM}{CC/SM}(SA)$	$\frac{1.59}{1.56}$(1.71)	$\frac{\mathbf{3.95}}{\mathbf{3.95}}$(3.86)	$\frac{SPC/ScM}{CC/ScM}$	$\frac{1.59}{\mathbf{1.50}}$	$\frac{3.92}{3.91}$
	HePy	$\frac{SPC/SM}{CC/SM}(SA)$	$\frac{1.69}{1.72}$(**1.63**)	$\frac{4.19}{4.35}$(3.79)	$\frac{SPC/ScM}{CC/ScM}$	$\frac{1.67}{1.66}$	$\frac{4.24}{\mathbf{4.40}}$
	YeMi	$\frac{SPC/SM}{CC/SM}(SA)$	$\frac{1.55}{1.61}$(1.54)	$\frac{4.11}{4.28}$(3.91)	$\frac{SPC/ScM}{CC/ScM}$	$\frac{\mathbf{1.53}}{1.60}$	$\frac{4.12}{\mathbf{4.33}}$
	AgPh	$\frac{SPC/SM}{CC/SM}(SA)$	$\frac{1.53}{\mathbf{1.50}}$(1.70)	$\frac{\mathbf{4.04}}{3.99}$(3.81)	$\frac{SPC/ScM}{CC/ScM}$	$\frac{1.55}{1.51}$	$\frac{3.97}{\mathbf{4.04}}$
	BuEb	$\frac{SPC/SM}{CC/SM}(SA)$	$\frac{1.65}{1.61}$(1.65)	$\frac{3.98}{\mathbf{4.09}}$(3.80)	$\frac{SPC/ScM}{CC/ScM}$	$\frac{1.68}{\mathbf{1.56}}$	$\frac{3.99}{4.08}$
DS2	Human Genome	$\frac{SPC/SM}{CC/SM}(SA)$	$\frac{1.40}{1.41}$(1.40)	$\frac{4.21}{\mathbf{4.31}}$(3.95)	$\frac{SPC/ScM}{CC/ScM}$	$\frac{1.42}{\mathbf{1.39}}$	$\frac{4.19}{4.18}$
DS3	E.Coli K-12	$\frac{SPC/SM}{CC/SM}(SA)$	$\frac{1.75}{\mathbf{1.72}}$(1.81)	$\frac{4.22}{\mathbf{4.24}}$(3.87)	$\frac{SPC/ScM}{CC/ScM}$	$\frac{1.79}{1.80}$	$\frac{4.10}{4.03}$
DS4	Isolates	$\frac{SPC/SM}{CC/SM}(SA)$	$\frac{1.68}{1.69}$(1.76)	$\frac{4.71}{\mathbf{4.84}}$(4.18)	$\frac{SPC/ScM}{CC/ScM}$	$\frac{\mathbf{1.65}}{1.71}$	$\frac{4.53}{4.72}$
Average of DS1		$\frac{SPC/SM}{CC/SM}(SA)$	$\frac{1.59}{1.58}$(**1.55**)	$\frac{4.02}{\mathbf{4.10}}$(3.93)	$\frac{SPC/ScM}{CC/ScM}$	$\frac{1.58}{1.56}$	$\frac{4.03}{4.09}$
Overall average		$\frac{SPC/SM}{CC/SM}(SA)$	$\frac{1.59}{1.58}$(**1.54**)	$\frac{4.08}{\mathbf{4.16}}$(3.94)	$\frac{SPC/ScM}{CC/ScM}$	$\frac{1.58}{1.57}$	$\frac{4.07}{4.12}$

The benchmarking results of HMG-GA(CC/SM) and previous compressors are shown in Fig-

ure 4. The performance of six compression methods is evaluated across four datasets, based on two compression metrics: CR and BPB. Higher (lower) CR (BPB) indicates better compression efficiency. For some compressors, either the CR or BPB values were unavailable. Specifically, NUHT terminated prematurely on datasets DS2 and DS4 due to runtime issues, while JARVIS3 encountered an “exec format error" during execution, preventing it from processing datasets DS2, DS3, and DS4. Additionally, NUHT, BZ2, and JARVIS3 did not provide BPB values for certain datasets, limiting a direct comparison.

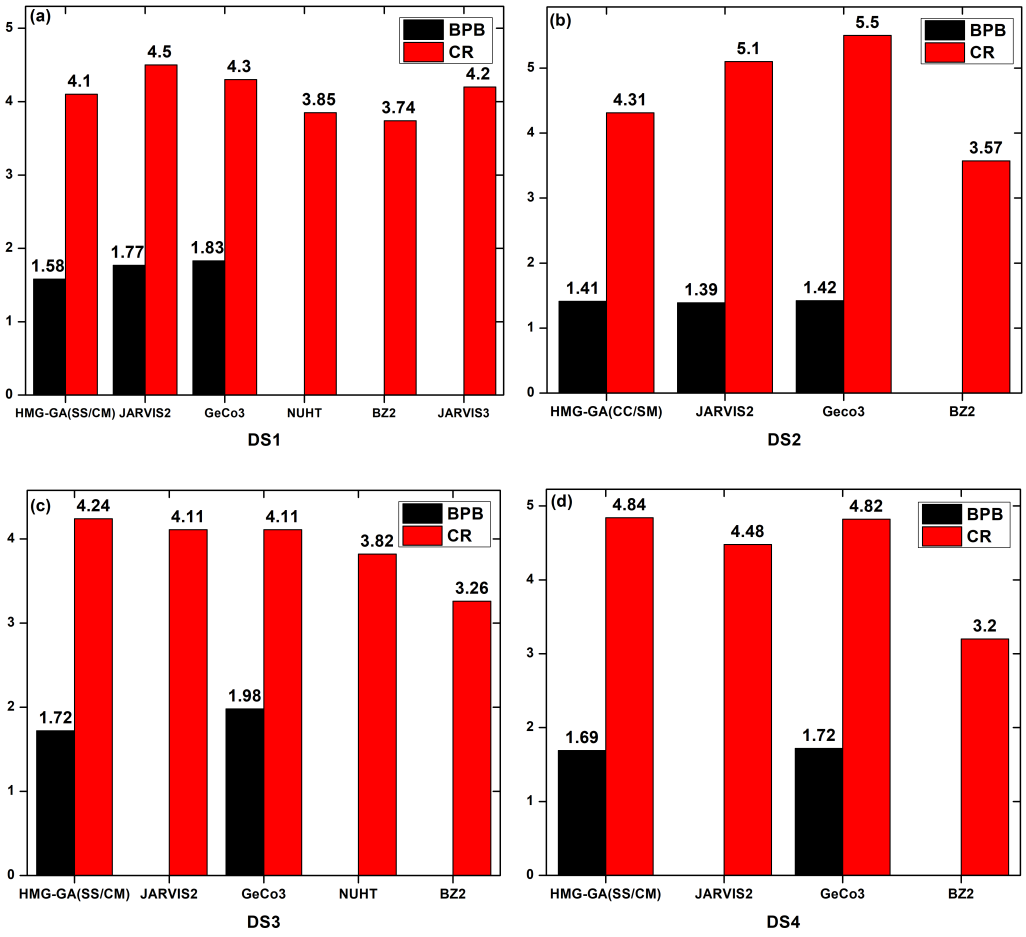


Figure 4: Compression benchmark in terms of BPB and CR for six compressors across four datasets. The HMG-GA(CC/SM) variant is used for benchmarking as it performed slightly better than HMG-SA and other variants of HMG-GA

HMG-GA(CC/SM) achieves the best BPB results across all datasets compared to other compressors, except for DS2, where its performance is slightly outperformed by JARVIS2. Specifically, for DS1, HMG-GA(CC/SM) provides 12% better compression than JARVIS2 and 15.8% better compression than GeCo3. On DS2, however, JARVIS2 achieves slightly better compression, outperforming HMG-GA(CC/SM) by 1.4%, while HMG-GA(CC/SM) surpasses GeCo3

by 0.8%. This highlights that dataset characteristics impact the relative performance of compression algorithms. For DS3 and DS4, HMG-GA(CC/SM) achieves slightly higher BPB compared to DS1 and DS2, indicating that these datasets may be more challenging to compress due to reduced redundancy or unique sequence patterns. Despite this, HMG-GA(CC/SM) still outperforms other methods on DS3 and DS4 in terms of BPB. For CR, HMG-GA(CC/SM) performed better than JARVIS2 (DS3, DS4), GeCo3 (DS3, DS4), BZ2 (DS1, DS3, DS4), NUHT (DS1, DS3). However, JARVIS3 has better CR than HMG-GA(CC/SM) on DS1.

Table 3: HMG results for compression time (in seconds) on four datasets

Dataset		C&M	Time	C&M	Time
DS1	HoSa	$\frac{SPC/SM}{CC/SM}(SA)$	$\frac{\mathbf{112.40}}{117.34}(125.00)$	$\frac{SPC/ScM}{CC/ScM}$	$\frac{121.01}{120.07}$
	GaGa	$\frac{SPC/SM}{CC/SM}(SA)$	$\frac{\mathbf{90.65}}{91.49}(96.11)$	$\frac{SPC/ScM}{CC/ScM}$	$\frac{92.10}{93.04}$
	DaRe	$\frac{SPC/SM}{CC/SM}(SA)$	$\frac{\mathbf{39.41}}{40.01}(43.40)$	$\frac{SPC/ScM}{CC/ScM}$	$\frac{39.46}{40.10}$
	OrSa	$\frac{SPC/SM}{CC/SM}(SA)$	$\frac{\mathbf{24.80}}{25.71}(32.30)$	$\frac{SPC/ScM}{CC/ScM}$	$\frac{25.01}{25.20}$
	DrMe	$\frac{SPC/SM}{CC/SM}(SA)$	$\frac{18.10}{\mathbf{17.63}}(22.30)$	$\frac{SPC/ScM}{CC/ScM}$	$\frac{18.51}{18.40}$
	EnIn	$\frac{SPC/SM}{CC/SM}(SA)$	$\frac{15.90}{\mathbf{15.85}}(18.20)$	$\frac{SPC/ScM}{CC/ScM}$	$\frac{16.02}{15.98}$
	ScPo	$\frac{SPC/SM}{CC/SM}(SA)$	$\frac{\mathbf{6.19}}{6.93}(7.50)$	$\frac{SPC/ScM}{CC/ScM}$	$\frac{6.68}{6.29}$
	PlFa	$\frac{SPC/SM}{CC/SM}(SA)$	$\frac{5.36}{5.40}(\mathbf{5.01})$	$\frac{SPC/ScM}{CC/ScM}$	$\frac{5.35}{5.51}$
	EsCo	$\frac{SPC/SM}{CC/SM}(SA)$	$\frac{\mathbf{2.88}}{2.90}(3.80)$	$\frac{SPC/ScM}{CC/ScM}$	$\frac{2.98}{3.01}$
	HaHi	$\frac{SPC/SM}{CC/SM}(SA)$	$\frac{2.10}{\mathbf{2.09}}(2.40)$	$\frac{SPC/ScM}{CC/ScM}$	$\frac{2.14}{2.12}$
	AeCa	$\frac{SPC/SM}{CC/SM}(SA)$	$\frac{0.92}{\mathbf{0.91}}(0.99)$	$\frac{SPC/ScM}{CC/ScM}$	$\frac{0.94}{0.92}$
	HePy	$\frac{SPC/SM}{CC/SM}(SA)$	$\frac{\mathbf{0.90}}{0.93}(0.98)$	$\frac{SPC/ScM}{CC/ScM}$	$\frac{\mathbf{0.90}}{0.91}$
	YeMi	$\frac{SPC/SM}{CC/SM}(SA)$	$\frac{0.14}{\mathbf{0.13}}(0.23)$	$\frac{SPC/ScM}{CC/ScM}$	$\frac{0.16}{0.20}$
	AgPh	$\frac{SPC/SM}{CC/SM}(SA)$	$\frac{0.09}{\mathbf{0.08}}(0.10)$	$\frac{SPC/ScM}{CC/ScM}$	$\frac{0.09}{0.09}$
	BuEb	$\frac{SPC/SM}{CC/SM}(SA)$	$\frac{0.07}{\mathbf{0.06}}(0.09)$	$\frac{SPC/ScM}{CC/ScM}$	$\frac{0.07}{0.08}$
DS2	Human Genome	$\frac{SPC/SM}{CC/SM}(SA)$	$\frac{\mathbf{2330}}{2452}(2781)$	$\frac{SPC/ScM}{CC/ScM}$	$\frac{2491}{2501}$
DS3	E.Coli K-12	$\frac{SPC/SM}{CC/SM}(SA)$	$\frac{9.05}{9.81}(\mathbf{8.80})$	$\frac{SPC/ScM}{CC/ScM}$	$\frac{9.10}{10.01}$
DS4	Isolates	$\frac{SPC/SM}{CC/SM}(SA)$	$\frac{434.65}{\mathbf{430.24}}(434.65)$	$\frac{SPC/ScM}{CC/ScM}$	$\frac{430.53}{441.49}$
Average of DS1		$\frac{SPC/SM}{CC/SM}(SA)$	$\frac{\mathbf{21.32}}{21.83}(23.89)$	$\frac{SPC/ScM}{CC/ScM}$	$\frac{22.09}{22.12}$
Overall average		$\frac{SPC/SM}{CC/SM}(SA)$	$\frac{616.74}{608.84}(\mathbf{595.01})$	$\frac{SPC/ScM}{CC/ScM}$	$\frac{613.49}{625.68}$

5.1.2. Compression and Decompression Time

The compression and decompression time for HMG-GA variants and HMG-SA are presented in Tables 3 and 4, respectively. For compression, on DS1, the SPC/SM variant of HMG-GA takes, on average, 21.32 seconds, while the HMG-GA(SPC/ScM) variant takes about 22.09 seconds. The HMG-GA(CC/SM) variant is reported at 21.83 seconds, with the HMG-GA(CC/ScM) taking about 22.12 seconds. The differences between these varinats are minimal, with SM methods generally performing slightly better than ScM with the only exception in HePy sequence of DS1, where ScM mutation method is quickest. Compression time for specific methods within DS1 varies, with times ranging from 5.36 seconds (PlFa using HMG-GA(SPC/SM)) to 112.40 seconds (HoSa using HMG-GA(SPC/SM)). The highest compression times tend to occur for datasets with more complex structures, while smaller datasets result in faster times. For DS2, the compression time for the HMG-GA(SPC/SM) is 2,330 seconds, which is the quickest among all the HMG methods. Similarly, for DS3, the HMG-SA method is the quickest, taking 8.80

seconds, and for DS4, the HMG-GA(CC/SM) is the fastest, taking 430.24 seconds. Overall, on average, the HMG-SA method is the quickest, 1.05 times faster than the HMG-GA(CC/ScM). One possible reason for HMG-SA faster performance is that it involves fewer steps, as there is no crossover operation in SA.

Table 4: HMG results for decompression time (in seconds) on four datasets

Dataset		C&M	Time	C&M	Time
DS1	HoSa	$\frac{SPC/SM}{CC/SM}(SA)$	$\frac{76.43}{75.44}$(**73.55**)	$\frac{SPC/ScM}{CC/ScM}$	$\frac{74.74}{79.51}$
	GaGa	$\frac{SPC/SM}{CC/SM}(SA)$	$\frac{58.00}{57.64}$(**56.41**)	$\frac{SPC/ScM}{CC/ScM}$	$\frac{60.51}{61.56}$
	DaRe	$\frac{SPC/SM}{CC/SM}(SA)$	$\frac{55.64}{56.88}$(58.31)	$\frac{SPC/ScM}{CC/ScM}$	$\frac{54.43}{\mathbf{52.09}}$
	OrSa	$\frac{SPC/SM}{CC/SM}(SA)$	$\frac{17.46}{18.51}$(17.99)	$\frac{SPC/ScM}{CC/ScM}$	$\frac{\mathbf{17.31}}{16.88}$
	DrMe	$\frac{SPC/SM}{CC/SM}(SA)$	$\frac{13.17}{14.51}$(15.11)	$\frac{SPC/ScM}{CC/ScM}$	$\frac{\mathbf{13.04}}{16.84}$
	EnIn	$\frac{SPC/SM}{CC/SM}(SA)$	$\frac{11.33}{11.84}$(12.04)	$\frac{SPC/ScM}{CC/ScM}$	$\frac{\mathbf{10.66}}{13.51}$
	ScPo	$\frac{SPC/SM}{CC/SM}(SA)$	$\frac{\mathbf{4.48}}{5.01}$(5.31)	$\frac{SPC/ScM}{CC/ScM}$	$\frac{5.85}{5.95}$
	PlFa	$\frac{SPC/SM}{CC/SM}(SA)$	$\frac{4.13}{4.24}$(5.69)	$\frac{SPC/ScM}{CC/ScM}$	$\frac{\mathbf{3.85}}{5.33}$
	EsCo	$\frac{SPC/SM}{CC/SM}(SA)$	$\frac{\mathbf{1.5}}{1.55}$(2.41)	$\frac{SPC/ScM}{CC/ScM}$	$\frac{1.83}{1.79}$
	HaHi	$\frac{SPC/SM}{CC/SM}(SA)$	$\frac{\mathbf{2.1}}{2.14}$(2.99)	$\frac{SPC/ScM}{CC/ScM}$	$\frac{3.44}{3.5}$
	AeCa	$\frac{SPC/SM}{CC/SM}(SA)$	$\frac{\mathbf{1.38}}{1.41}$(1.53)	$\frac{SPC/ScM}{CC/ScM}$	$\frac{1.55}{1.65}$
	HePy	$\frac{SPC/SM}{CC/SM}(SA)$	$\frac{\mathbf{1.15}}{1.23}$(1.29)	$\frac{SPC/ScM}{CC/ScM}$	$\frac{1.22}{1.31}$
	YeMi	$\frac{SPC/SM}{CC/SM}(SA)$	$\frac{\mathbf{0.15}}{0.19}$(0.21)	$\frac{SPC/ScM}{CC/ScM}$	$\frac{0.20}{0.22}$
	AgPh	$\frac{SPC/SM}{CC/SM}(SA)$	$\frac{\mathbf{0.1}}{\mathbf{0.1}}$(0.15)	$\frac{SPC/ScM}{CC/ScM}$	$\frac{0.14}{0.19}$
	BuEb	$\frac{SPC/SM}{CC/SM}(SA)$	$\frac{\mathbf{0.03}}{\mathbf{0.03}}$(0.05)	$\frac{SPC/ScM}{CC/ScM}$	$\frac{\mathbf{0.03}}{0.04}$
DS2	Human Genome	$\frac{SPC/SM}{CC/SM}(SA)$	$\frac{605.99}{\mathbf{595.63}}$(611.45)	$\frac{SPC/ScM}{CC/ScM}$	$\frac{608.88}{601.84}$
DS3	E.Coli K-12	$\frac{SPC/SM}{CC/SM}(SA)$	$\frac{10.81}{9.84}$(10.41)	$\frac{SPC/ScM}{CC/ScM}$	$\frac{\mathbf{9.15}}{10.99}$
DS4	Isolates	$\frac{SPC/SM}{CC/SM}(SA)$	$\frac{434.65}{\mathbf{430.24}}$(434.65)	$\frac{SPC/ScM}{CC/ScM}$	$\frac{430.53}{441.49}$
Average of DS1		$\frac{SPC/SM}{CC/SM}(SA)$	$\frac{\mathbf{16.47}}{16.71}$(16.86)	$\frac{SPC/ScM}{CC/ScM}$	$\frac{16.58}{17.35}$
Overall average		$\frac{SPC/SM}{CC/SM}(SA)$	$\frac{72.13}{\mathbf{71.46}}$(72.75)	$\frac{SPC/ScM}{CC/ScM}$	$\frac{72.07}{73.03}$

For DS1, the decompression time for the HMG-GA(SPC/SM) is, on average, 16.47 seconds, while the HMG-GA(SPC/ScM) takes about 16.58 seconds. Among the tested variants, the HMG-GA(CC/SM) takes 16.71 seconds on average, while the HMG-GA(CC/ScM) takes about 17.35 seconds. The decompression time differences between HMG-GA varinats are minimal, with the SM methods generally being slightly faster than ScM. Decompression time for specific HMG-GA variants within DS1 varies, ranging from 0.03 seconds (BuEb using HMG-GA(SPC/SM)) to 76.43 seconds (HoSa using HMG-GA(CPC/SM)). The higher decompression times tend to occur for larger datasets, such as the Human Genome, while smaller datasets result in faster times. For DS2, the decompression time for the HMG-GA(CC/SM) is 595.63 seconds, which is the fastest, followed by HMG-GA(CC/ScM) at 601.84 seconds. HMG-GA(SPC/ScM) takes about 608.88 seconds and HMG-GA(SPC/SM) takes 605.99 seconds. For DS3, the HMG-GA(SPC/ScM) is the fastest, taking 9.15 seconds, followed by HMG-GA(CC/SM) at 9.84 seconds, while HMG-GA(SPC/SM) takes 10.81 seconds and HMG-GA(CC/ScM) takes 10.99 seconds. For DS4, the decompression time for HMG-GA(CC/SM) is the fastest at 430.24 seconds, followed closely by HMG-GA(SPC/ScM) at 430.53 seconds. The HMG-GA(SPC/SM) takes about 434.65 seconds, while HMG-GA(CC/ScM) takes 441.49 seconds. For the overall average, the HMG-GA(SPC/SM) takes 72.13 seconds, HMG-GA(CC/SM) takes 71.46 seconds and

HMG-SA takes 72.75 seconds, with HMG-GA(SPC/ScM) and HMG-GA(CC/ScM) showing similar performance at 72.07 seconds and 73.03 seconds, respectively.

Figure 5 presents the compression and decompression times (in seconds). The results are split into two sections: the compressors to the left of the blue vertical line are those that produced compression and decompression times for all four datasets. In contrast, the compressors to the right of the blue vertical line generated results for a subset of datasets (JARVIS3 and NUHT for DS1 only). It can be observed that the general-purpose compressor, BZ2, is the fastest across all four datasets, but it delivers less effective compression results. In comparison, HMG-GA(CC/SM) outperforms JARVIS2 and GeCo3 in both compression and decompression tasks. Specifically, it is 2.35 and 2.80 times faster in compression and 5.82 and 1.48 times faster in decompression compared to JARVIS2 and GeCo3, respectively. Among the methods that provide results for a subset of datasets, JARVIS3 emerges as the fastest, followed by NUHT.

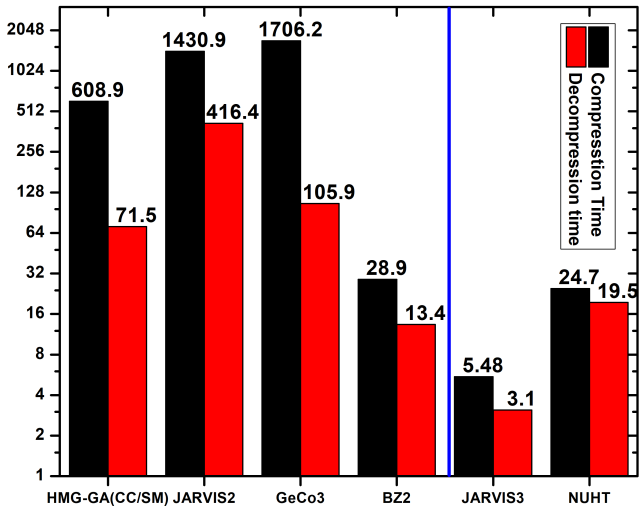


Figure 5: Compression and decompression times (in seconds) benchmarked for six compressors across four datasets. Note that the y-axis is displayed on a log scale with base 2

5.2. Convergence

An additional experiment is conducted using a sequence from DS1 (HoSa) and DS2 across all variants of HMG-GA and HMG-SA. This experiment aimed to observe HMG's convergence behavior, specifically how the compression ratio evolves over successive iterations and how k-mer selection improves during this process. The convergence of the compression ratio was evaluated based on two key factors: (1) the number of k-mers added to the KT and (2) the number of generations required to evolve a set of k-mers before they are incorporated into the KT. This experiment provides valuable insight into how HMG refines its approach to compression by iteratively adjusting the k-mer selection, leading to improved compression performance.

The results are presented in Figure 6. The solid lines represent the results of HMG for the DS1 sequence, while the dotted lines depict the results for DS2. The bottom x-axis illustrates the cumulative number of k-mers added to the KT, while the top x-axis represents the number of generations that HMG uses to evolve each k-mer set. The y-axis displays the compression

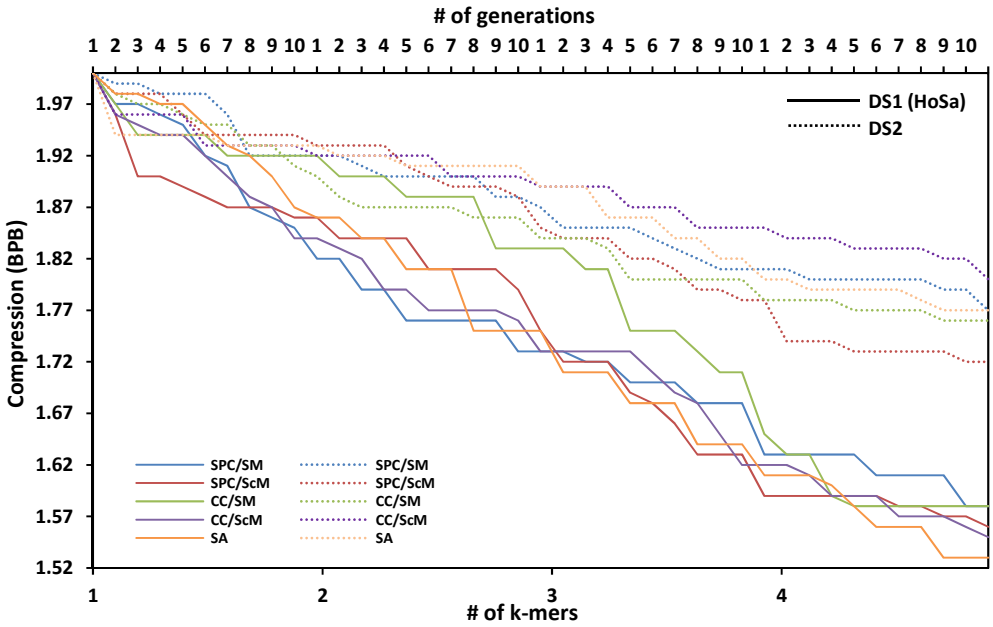


Figure 6: Comparison of compression with respect to number of k-mers and generations

ratio, which indicates the algorithm's effectiveness in reducing data size. For clarity, the number of sequences are considered within the ranges of [1, 4]. For the HMG-GA variants, the number of generations is fixed at 10 for each k-mer. In the case of HMG-SA, the temperature is initially set to 100 and gradually reduced by multiplying it by a cooling rate of 0.95. The loop terminates when the temperature drops below 1. For consistency, the temperature decrease from 100 to 1 is assumed to occur over the same 10 generations as used in HMG-GA.

In the figure, the BPB of HMG-GA variants and HMG-SA exhibit a common pattern: an initial high BPB that improves as more k-mers are added. This indicates that the initial k-mer added to the KT has a substantial impact on improving compression, with subsequent k-mers have trivial gains. The HMG-GA variants and HMG-SA obtained better BPB on HoSa as compared to DS2. HMG-GA(CC/SM) (solid) performance gets better after the eighth generation of the second k-mer. HMG-GA(SPC/SM) (solid) achieved very low BPB in the first seven generations of the first k-mer. After that its performance is steady for the rest of the generations.

It is noteworthy that the Human Genome in DS2 and Isolates in DS4 utilized more k-mers (8 in total) than the other sequences in DS1 and DS3 (4 in total). This means that genome sequences of DS2 and DS4 required more k-mers for better representation and compression. As discussed in Section 4.1, HMG is strategically designed to either improve the compression or maintain its current quality, ensuring no degradation over time. Figure 6 illustrates this behavior for HoSa of DS1 and DS2, showing that compression ratios consistently decrease or remain stable as the number of generations and k-mers increases. This trend highlights HMG's effectiveness in progressively refining or sustaining data compression efficiency through iterative optimization.

5.3. Classification

In another experiment, we assessed the quality of the optimal k-mers from the *KT*, obtained with HMG, by employing them as features to train a classification model. The Weka [60] software was used for training/testing six classifiers, which are Naive Bayes (NB), Logistic Regression (LR), Support Vector Machine (SVM), k-nearest Neighbor (kNN), Decision Tree (DT), and Random Forest (RF). These classifiers were evaluated using their default hyperparameters and a 10-fold cross-validation. The effectiveness of the classifiers was evaluated using the accuracy metric, which represents the percentage of correctly classified instances based on the optimal k-mers. The obtained accuracies by the six classifiers are listed in Table 5.

Table 5: Classifiers accuracy (in %) on optimal k-mers

Dataset↓, Classifier→		**NB**	**LR**	**SVM**	**kNN**	**DT**	**RF**
DS1	HoSa	91.25	**95**	**95**	88.75	**95**	**95**
	GaGa	60	**95**	**95**	87.5	**95**	90
	DaRe	93.75	95	95	90	95	**96.25**
	OrSa	50	**95**	**95**	80	**95**	91.25
	DrMe	90	91.25	**95**	88.75	**95**	92.5
	EnIn	87.5	88.75	**95**	87.5	**95**	93.75
	ScPo	90	**95**	**95**	86.25	**95**	93.75
	PlFa	81.25	92.5	**95**	88.75	**95**	92.5
	EsCo	92.5	93.75	**95**	87.25	**95**	93.75
	HaHi	73.75	93.75	**95**	92.5	**95**	93.75
	AeCa	92.5	92.5	**95**	90	**95**	92.5
	HePy	92.5	**95**	**95**	91.25	**95**	93.75
	YeMi	88.75	92.5	**95**	88.75	**95**	92.5
	AgPh	70	**95**	**95**	87.5	**95**	90
	BuEb	93.75	95	95	90	95	**96.25**
DS2	Human Genome	83.75	86.25	90	83.75	90	86.25
DS3	E.coli K-12	70	88.75	**90**	73.75	**90**	83.75
DS4	Isolates	91.25	**95**	**95**	90	**95**	**95**
Overall average		82.91	93.05	**94.44**	87.34	**94.44**	92.36

SVM and DT performed similarly on all datasets, with an average accuracy of 94.44%, followed by LR (93.05)% and RF (92.36%). In contrast, previous studies [32, 33, 34] employed compression measures like NC, along with other features, for the taxonomic classification of metagenomic, Archaea and viruses, respectively. XGBoost demonstrated the best performance on the given features, achieving average accuracies of 95% [32], 92.10% [33] and 82.15% [34]. In this paper, the optimal k-mers enhance the interpretability of HMG by revealing the distribution and frequency of k-mers that significantly contribute to the compression of genome sequences.

In addition to the standard classifier, a custom classifier is developed to classify k-mers for multiple datasets, each stored in separate files. The k-mers are generated using variants of HMG-GA and HMG-SA, which are employed to explore the search space and identify optimal k-mers. The classifier first processes these files by counting the occurrences of each k-mer in all datasets.

For each k-mer, the classifier then determines how many times it appears in each dataset, accounting for overlapping occurrences within the dataset. To ensure fairness in classification and prevent larger datasets from having an advantage, the occurrences are normalized with respect to the size of each dataset. This normalization is achieved by calculating the frequency of each k-mer as a percentage of the dataset size, which ensures that the frequency count reflects the proportion of the dataset rather than its absolute size. As a result, larger datasets cannot bias the classification process simply due to having more entries.

After calculating the normalized occurrences, the classifier assigns a predicted class to each k-mer based on the dataset with the highest normalized frequency. For instance, if a k-mer occurs most frequently in DS1, it will be classified as belonging to DS1. Once the predictions are made, the classifier compares them to the actual class for each k-mer, which is derived from the k-mer file name (e.g., a file named BuEb corresponds to the class BuEb). The classifier then calculates the accuracy by determining the percentage of correct classifications. The accuracy of these predictions serves as the primary measure of the classifier's performance, providing a useful tool for classifying genomic or biological sequence data based on optimal k-mer content.

More formally, the process can be described as follows: Let D_i represent the i-th dataset, where $i \in \{1, 2, \ldots, N\}$ and N is the total number of datasets. For a given k-mer, kM_m, the frequency of kM_m in dataset D_i is denoted as $f(kM_m, D_i)$. The size of dataset D_i, represented by $|D_i|$, is used to normalize the frequency of kM_m within that dataset, giving the normalized frequency $f_{\text{norm}}(kM_m, D_i)$, calculated as:

$$f_{\text{norm}}(kM_m, D_i) = \frac{f(kM_m, D_i)}{|D_i|}$$

The classifier assigns a predicted class $\hat{C}(kM_m)$ to the kM_m by selecting the dataset D_i where the normalized frequency $f_{\text{norm}}(kM_m, D_i)$ is the highest, expressed as:

$$\hat{C}(kM_m) = \arg\max_i \left(f_{\text{norm}}(kM_m, D_i)\right)$$

The predicted class $\hat{C}(kM_m)$ is then compared to the actual class $C(kM_m)$, which is derived from the k-mer's. The classifier's accuracy ACC is computed by determining the percentage of correctly classified k-mers, using the indicator function $\mathbb{I}(\hat{C}(kM_m) = C(kM_m))$, as follows:

$$ACC = \frac{1}{M} \sum_{m=1}^{M} \mathbb{I}(\hat{C}(kM_m) = C(kM_m))$$

where M is the total number of k-mers, and $\mathbb{I}(\hat{C}(kM_m) = C(kM_m))$ equals 1 if the predicted class matches the actual class, and 0 otherwise. The final accuracy ACC represents the percentage of correctly classified k-mers out of the total M evaluated.

The accuracies achieved by each variant of the HMG-GA and HMG-SA are presented in Table 6. The results show that the custom classifier with HMG-SA achieved the highest accuracy (93.22%) on average, followed by HMG-GA(CC/SM) (93.07%). The HMG-GA(SPC/ScM) performed better than HMG-GA(SPC/SM) and HMG-GA(CC/ScM), with an accuracy of 92.80% compared to 92.37% and 91.45%.

In summary, the results indicate that HMG outperforms in terms of compression (BPB) and runtime. Additionally, the effectiveness of HMG as a descriptor for data, specifically genome sequences, is apparent in two classification scenarios. These findings establish HMG as an optimal solution for handling genomic data, offering advantages not only in terms of data storage and

Table 6: Classification accuracies (in %) of k-mers based on the custom classifier

Dataset		SPC/SM	SPC/ScM	CC/SM	CC/ScM	SA
DS1	HoSa	94.91	**95.10**	93.22	88.23	94.11
	GaGa	93.17	92.42	**94.11**	91.23	94.02
	DaRe	91.65	**94.41**	92.67	92.34	93.01
	OrSa	92.42	93.89	**94.60**	90.75	93.59
	DrMe	**94.11**	91.50	93.66	90.83	93.45
	EnIn	91.34	93.17	92.56	90.32	**94.34**
	ScPo	92.56	**94.34**	93.71	91.19	92.87
	PlFa	92.11	91.76	93.34	90.89	**94.02**
	EsCo	90.65	93.02	**94.13**	92.78	91.76
	HaHi	**94.01**	91.23	92.89	90.44	93.65
	AeCa	91.49	92.83	**94.21**	91.08	93.67
	HePy	90.89	**94.45**	93.33	91.65	93.11
	YeMi	**94.13**	92.67	90.98	92.42	93.47
	AgPh	90.25	91.77	92.88	**94.02**	91.34
	BuEb	91.55	**94.67**	91.98	92.65	93.02
DS2	Human Genome	93.51	91.32	94.23	**96.47**	93.99
DS3	E.Coli	90.33	89.94	**91.10**	88.01	90.80
DS4	Isoaltes	93.61	92.09	91.78	90.90	**93.81**
Overall average		92.37	92.80	93.07	91.45	**93.22**

processing speed but also facilitates genome classification, making it suitable for a wide range of applications in genomic research and bioinformatics workflows.

6. Conclusion

This paper introduced a novel framework, called HMG, that integrated heuristic algorithms with the MDL principle for efficient reference-free lossless compression of genomic data. The two algorithms, named HMG-GA and HMG-SA, are based on fusing the GA and SA, respectivley, with the MDL principle. HMG effectively captured the sequential structure of genome sequences and derived optimal k-mers played critical part in the compression. Experimental comparison with state-of-the-art genome compressors has shown that HMG is fast, and provides a low BPB. The principle of MDL ensures that HMG 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. HMG's potential extends beyond compression; it offers a multi-faceted advantage by utilizing the identified k-mers for genome classification, thereby enhancing interpretability and utility in genome compression research. As genomic data continues to expand, HMG provides cost-effective storage solutions and enables faster, more streamlined analysis workflows in both research and clinical settings. HMG is freely available as open-source, thereby guaranteeing accessibility and promoting collaboration within the scientific community.

Despite HMG's demonstrated performance improvements, several limitations and areas for future research remain. Firstly, the effectiveness of the k-mer selection process is highly dependent on the quality of the heuristics, which might not always guarantee global optimality.

Secondly, while HMG performs well on the benchmark datasets, its generalizability to a wider range of genomic and metagenomic datasets has not yet been fully explored, and further validation across diverse data types is required. Thirdly, the multi-step encoding process introduced, particularly for large datasets, which could be mitigated by integrating GPU acceleration or parallel processing techniques. Fourthly, the employed encoding schemes, such as Huffman and substitution encoding, may not be optimal for all genomic datasets, and exploring more advanced techniques like arithmetic encoding or range encoding could further enhance compression efficiency. Fifthly, the effectiveness of HMG could be improved by incorporating more sophisticated techniques for pattern recognition, such as palindrome matching or sequence alignment, to identify gaps and further optimize compression. incorporating 'gaps' between bases during compression, complemented by techniques such as palindrome matching, repeat detection, and sequence alignment, to enhance pattern identification and achieve greater compression efficiency. Lastly, adding more measures, such as NC, to HMG will facilitate more compression-based analysis.

Funding: This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

CRediT author statement

M. Zohaib Nawaz: Software, Data Curation, Methodology, Validation, Visualization, Writing - Original Draft, Writing - Review & Editing. **M. Saqib Nawaz**: Formal Analysis, Validation, Investigation, Visualization, Methodology, Writing - Original Draft, Writing - Review & Editing. **Philippe Fournier-Viger:** Supervision, Resources, Conceptualization, Methodology, Validation, Investigation, Writing - Review & Editing. **M. Shoaib Nawaz:** Data Curation, Validation, Writing - Review & Editing. **Jerry Chun-Wei Lin:** Investigation, Writing - Review & Editing. **Vincent S. Tseng:** Validation, Writing - Review & Editing.

Conflict of Interest: Authors declare no conflict of interest.

References

[1] Q. Li, Z. Xiong, P. Xiang, L. Zhou, T. Zhang, Q. Wu, C. Zhao, Effects of uranium mining on soil bacterial communities and functions in the Qinghai-Tibet plateau, Chemosphere, 347 (2024) 140715.

[2] M. J. Van Oppen, M. A. Coleman, Advancing the protection of marine life through genomics, PLoS Biology, 20 (10) (2022) e3001801.

[3] H. L. Huang, S. Yin, H. Zhao, C. Tian, J. Huang, S. Deng, Z. Li, DNA extraction and sequencing of the Mawangdui ancient cadaver protected by formalin, bioRxiv, (2020).

[4] S. Mammola, I. R. Amorim, M. E. Bichuette, P. A. V. Borges, N. Cheeptham, S. J. B. Cooper, D. C. Culver, et al., Fundamental research questions in subterranean biology, Biological Reviews, 95 (6) (2020) 1855-1872.

[5] L. H. Rieseberg, Chromosomal rearrangements and speciation, Trends in Ecology & Evolution, 16 (7) (2001) 351-358.

[6] G. S. Roeder G. R. Fink, DNA rearrangements associated with a transposable element in yeast, Cell, 21, (1) (1980) 239-249.

[7] G. J. M. Zajac, L. G. Fritsche, J. S. Weinstock, S. L. Dagenais, R. H. Lyons, C. M. Brummett, G. R. Abecasis, Estimation of DNA contamination and its sources in genotyped samples, Genetic Epidemiology, 43 (8) (2019) 980-995.

[8] M. A. Barnes, C. R. Turner, C. L. Jerde, M. A. Renshaw, W. L. Chadderton, and D. M. Lodge, Environmental conditions influence eDNA persistence in aquatic systems, Environmental Science & Technology, 48 (3) (2014) 1819-1827.

[9] A. J. Adams, J. P. LaBonte, M. L. Ball, K. L. Richards-Hrdlicka, M. H. Toothman, C. J. Briggs, DNA extraction

method affects the detection of a fungal pathogen in formalin-fixed specimens using qPCR, PLoS One, 10 (8) (2015) e0135389.
[10] K. Slooten, The analogy between DNA kinship and DNA mixture evaluation, with applications for the interpretation of likelihood ratios produced by possibly imperfect models, Forensic Science International: Genetics, 52 (2021) 102449.
[11] M. Zitnik, F. Nguyen, B. Wang, J. Leskovec, A. Goldenberg, M. M. Hoffman, Machine learning for integrating data in biology and medicine: Principles, practice, and opportunities, Information Fusion, 50 (2019), 71-91.
[12] Z. D. Stephens, S. Y. Lee, F. Faghri et al., Big data: astronomical or genomical?, PLoS Biology, 13 (7) (2015) e1002195
[13] G. Berger, J. Peng, M. Singh, Computational solutions for omics data", Nature Review Genetics, 14 (5) (2013) 333–346.
[14] 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, Computers in Biology and Medicine, 183 (2024) 109307.
[15] D. Pratas, A. J. Pinho, JARVIS2: a data compressor for large genome sequences, in Proc. of DCC, (2023) 288–297.
[16] M. Silva, D. Pratas, A. J. Pinho, Efficient DNA sequence compression with neural networks, GigaScience, 9 (11) (2020) giaa119.
[17] S. Grumbach, F. Tahi, Compression of DNA sequences, in Proc. of DCC, (1993) 340–350.
[18] M. Hosseini, D. Pratas, A. J. Pinho, A survey on data compression methods for biological sequences, Information, 7 (4) (2016) 56.
[19] S. Kumar, M. P. Singh, S. R. Nayak, A. U. Khan, A. K. Jain, P. Singh, M. Diwakar, T. Soujanya, A new efficient referential genome compression technique for FastQ files", Funct. Integr. Genomics. vol. 23, no. 333, 2023.
[20] Z. Lu, L. Guo, J. Chen, Reference-based genome compression using the longest matched substrings with parallelization consideration, BMC Bioinformatics, 24 (369) (2023).
[21] S. Saha, S. Rajasekaran, ERGC: an efficient referential genome compression algorithm, Bioinformatics, 31 (21) (2015) 3468–3475.
[22] S. Wandelt, U. Leser, FRESCO: referential compression of highly similar sequences, IEEE/ACM Transactions on Computational Biology and Bioinformatics, 10 (5) (2013) 1275–1288.
[23] K. -O. Cheng, N. -F Law, W. -C Siu, Clustering-based compression for population DNA sequences, IEEE/ACM Transactions on Computational Biology and Bioinformatics, 16 (1) (2019) 208–221.
[24] S Roy, D. K. Maity, A. Mukhopadhyay, A lossless reference-free sequence compression algorithm leveraging grammatical, statistical, and substitution rules, Briefings in Functional Genomics, 24 (2025) elae050.
[25] M. J. P. Sousa, A. J. Pinho, D. Pratas, JARVIS3: an efficient encoder for genomic data, Bioinformatics, (2024) btae725.
[26] Z. Sun, Z., M. Wang, S. Kwong, LEC-Codec: Learning-based genome data compression, IEEE/ACM Transactions on Computational Biology and Bioinformatics, (2024) 1–12.
[27] S. K. Sheena, M. S. Nair, GenCoder: A novel convolutional neural network based autoencoder for genomic sequence, IEEE/ACM Transactions on Computational Biology and Bioinformatics, 21 (3) (2024) 405–415.
[28] A. Al-Okaily, B. Almarri, S. Al Yami, and C. H. Huang, Toward a better compression for DNA sequences using Huffman encoding, Journal of Computational Biology, 24 (4) (2017) 280-288.
[29] S. Alyami, C. -H Huang, Nongreedy unbalanced Huffman tree compressor for single and multifasta files, Journal of Computational Biology, 27 (6) (2020) 868–876.
[30] 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 of BIBM, (2018) 270–274.
[31] M. D. Cao, T. I. Dix, L. Allison, C. Mears, A simple statistical algorithm for biological sequence compression, in: Proc. of DCC, (2007) 43–52.
[32] J. M. Silva, J. R. Almeida, Enhancing metagenomic classification with compression-based features,, Artificial Intelligence in Medicien, 156 (2024) 102948.
[33] J. M. Silva, D. Pratas, T. Caetano, S. Matos, Feature-based classification of Archaeal sequences using compression-based methods, in Proc. of IbPRIA, (2022) 309-320.
[34] J. M. Silva, D. Pratas, T. Caetano, S. Matos, The complexity landscape of viral genomes", GigaScience, 11 (2022) giac079.
[35] S. Grabowski, T. M. Kowalski, MBGC: multiple bacteria genome compressor, GigaScience, 11 (2020) giab099.
[36] K. Kryukov, M. T. Ueda, S. Nakagawa, T. Imanishi, "Nucleotide archival format (NAF) enables efficient lossless reference-free compression of DNA sequences, Bioinformatics, 35 (19) (2019) 3826–3828.
[37] A. J. Pinho, D. Pratas, MFCompress: a compression tool for FASTA and multi-FASTA data, Bioinformaics, 30 (1) (2014) 117–118.
[38] 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", Bioinformatics, 28 (19) (2012) 2527–2529.
[39] K. S. Sheena, M. S. Nair, DNACoder: a CNN-LSTM attention-based network for genomic sequence data compres-

sion, Neural Computing and Applications, 36 (2024) 18363–18376.
[40] W. Cui, Z. Yu, Z., Liu, G. Wang, X. Liu, Compressing genomic sequences by using deep learning, in Proc. ofICANN, (2020), 92–104.
[41] J. H. Holland, Adaptation in natural and artificial systems, (1975), MIT Press.
[42] D. Bertsimas, J. Tsitsiklis, Simulated annealing, Statistical Science, 8 (1) (1993) 10-15.
[43] P. D. Grunwald, The Minimum Description Length Principle, (2007), MIT Press.
[44] M. Z. Nawaz, M. S. Nawaz, P. Fournier-Viger, V. S. Tseng, An MDL-based Genetic Algorithm for genome sequence compression, in Proc. of BIBM, (2024) 6724-6731.
[45] Q. Meng, S. Chandak, Y. Zhu, T. Weissman, Reference-free lossless compression of nanopore sequencing reads using an approximate assembly approach. Scientific Reports, 13 (2023) 2082.
[46] F. Nazari, S. Patel, Melissa LaRocca, Ryan Czarny, Giana Schena, Emma K. Murray. GeneSqueeze: A novel lossless, reference-Free compression algorithm for FASTQ/A files, bioRxiv, (2024), 2024.03.21.586111;
[47] S. Ozan, DNA sequence classification with compressors, arXiv preprint, (2024), arXiv:2401.14025.
[48] M. Gen, L. Lin, Genetic algorithms and their applications, in Handbook of Engineering Statistics, 674 (2022) 635-674, Springer.
[49] M. S. Nawaz, M. Z. Nawaz, O. Hasan, P. Fournier-Viger, M. Sun, An evolutionary/heuristic-based proof searching framework for interactive theorem prover, Applied Soft Computing, 104 (2021) 107200.
[50] D. Delahaye, S. Chaimatanan, M. Mongeau, Simulated annealing: From basics to applications, in: Handbook of Metaheuristics, 272 (2019) 1-35, Springer.
[51] B. Selman, C. P. Gomes, Hill-climbing search, Encyclopedia of Cognitive Science, (2006).
[52] M. S. Nawaz, P. Fournier-Viger, U. Yun, Y. Wu, W. Song, Mining high utility itemsets with Hill Climbing and Simulated Annealing, ACM Transactions on Management Information System, 13 (1) (2021) 1 - 22.
[53] M. S. Nawaz, P. Fournier-Viger, A. Shojaee, H. Fujita, Using artificial intelligence techniques for COVID-19 genome analysis, Applied Intelligence, 51 (2021) 3086-3103.
[54] J. Vreeken, M. van Leeuwen, A. Siebes, KRIMP: mining itemsets that compress, Data Mining and Knowledge Discovery, 23 (2011) 169-214.
[55] 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, Applied Intelligence 51 (2021) 1580-1601.
[56] D. Pratas, A. J. Pinho, A DNA sequence corpus for compression benchmark, in Proc. of PACBB, (2018) 208-215.
[57] S. Nurk, S. Koren, A. Rhie, et al., The complete sequence of a human genome, Science, 376 (6588) (2022) 44-53.
[58] J. Quick, N. J. Loman, Bacterial whole-genome read data from the Oxford Nanopore Technologies MinION nanopore sequencer, GigaScience Database, 2 (2014).
[59] P. S. Peter, M. Bosio, C. Gross, D. Bezdan, J. Gutierrez, P. Oberhettinger, J. Liese, W. Vogel, D. Dorfel, L. Berger, M. Marschal, Tracking of antibiotic resistance transfer and rapid plasmid evolution in a hospital setting by Nanopore sequencing, Msphere, 5 (4) (2020) 10-1128.
[60] 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.