2025 IEEE 31th International Conference on Parallel and Distributed Systems (ICPADS) | 979-8-3315-4901-5/25/$31.00 ©2025 IEEE | DOI: 10.1109/ICPADS67057.2025.11323030

GTree: A Trie-Based Alignment-Free Clustering Framework for Efficient DNA Data Storage and Enhanced Error Resilience

Linxuan Han*, Guanjin Qu*, Zihui Yan†‡ and Huaming Wu*‡
*Center for Applied Mathematics, Tianjin University, China
†School of Synthetic Biology and Biomanufacturing, Tianjin University, China
‡State Key Laboratory of Synthetic Biology, Tianjin University, China.
Emails: {hanlinxuan, quguanjin, yanzh, whming}@tju.edu.cn

***Abstract*—With the rapid advancements in information technologies, global data volumes have increased exponentially. Deoxyribonucleic acid (DNA)-based data storage has gained significant attention due to its superior storage capacity and long-term preservation potential compared to traditional media. However, during this process, DNA sequences may be affected by errors such as deletions, insertions, and substitutions, resulting in altered copies of the original sequence. Furthermore, the sequences in the DNA pool lack inherent order, preventing direct access to specific fragments. In this paper, we propose GTree, a trie-based DNA clustering framework tailored for DNA data storage. GTree achieves both high accuracy and low runtime complexity. It introduces a lightweight pre-screening module that combines Q-gram feature extraction, MinHash signatures, and Locality Sensitive Hashing (LSH) indexing to efficiently reduce the comparison space. A majority-label voting mechanism is also integrated to enhance clustering robustness in the presence of sequencing errors. Finally, we validate the performance of GTree through clustering experiments on both real biological and simulated datasets, demonstrating its superior performance.**

***Index Terms*—DNA-based data storage, DNA clustering, Sequencing errors, MinHash signatures**

I. Introduction

In recent years, with the advent of the information age and the vigorous development of Internet of Things (IoT), the application of large-scale IoT technology has led to the generation and collection of massive amounts of data [1]. The International Data Corporation (IDC) has predicted that the total amount of data generated worldwide is experiencing explosive growth, with an average annual increase of nearly 59%. According to this trend, the scale of data in 2025 may reach 180 zettabytes, and by 2035, this figure is even expected to exceed the magnitude of 1,000 zettabytes [2]. In the face of such enormous data pressure, traditional data storage methods—whether optical storage (e.g., optical discs), magnetic media (e.g., hard drives and tapes) or electronic devices (e.g., flash memory)—have exposed obvious bottlenecks in storage density, service life and energy consumption. Therefore, developing a new storage technology with breakthrough potential in key performance aspects such as density, durability, energy efficiency and security has become an urgent issue to be solved.

Deoxyribonucleic acid (DNA) is a compelling candidate for next-generation data storage. Built from a four-base alphabet of adenine (A), thymine (T), guanine (G), and cytosine (C), DNA acts as the genetic information carrier for all life. As depicted in Fig. 1, the DNA data storage workflow is a six-stage process: encoding, synthesis, preservation, acquisition, sequencing (reading), and decoding.

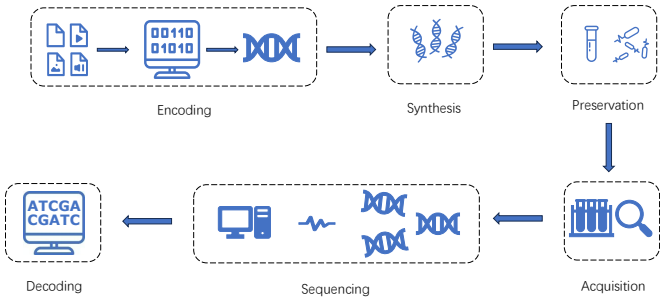

Fig. 1. DNA storage process

DNA has emerged as a promising solution for long-term information storage due to its high storage density, vast capacity, and extended preservation time. It holds the potential to revolutionize the way we store and preserve digital data [3], [4], [5], [6]. However, current synthesis and sequencing technologies, along with other biochemical factors, introduce errors such as base insertions, deletions, substitutions, and strand breaks during DNA synthesis and amplification. The occurrence probability of each error type varies, with the overall error rate being predominantly influenced by the synthesis and sequencing methods employed [7]. To effectively reconstruct the original sequences during sequencing and extract meaningful information, it is crucial to cluster the unordered distribution of a large number of DNA sequences. This clustering process plays a vital role in improving sequencing accuracy, particularly for data with high error rates, which directly affects the accuracy of subsequent information reduction and the overall efficiency of the system [8], [9], [10].

The clustering problem itself is not new, with numerous technically advanced methods developed long before the formal introduction of DNA storage technology. Consequently,

DNA sequence clustering can benefit from established clustering techniques, such as the k-means algorithm [11], mean shift clustering, DBSCAN [12], and hierarchical clustering. These methods have been successfully applied across various fields. However, DNA sequences have unique biological structures, meaning that treating DNA clustering as a standard text clustering problem is insufficient. That said, there are existing works that apply traditional clustering algorithms to DNA sequences. For example, MeShClust [13] employs the mean shift algorithm, commonly used in image processing and computer vision, while ADRS-CNet [14] applies the k-means technique during the final clustering stage. These approaches build upon traditional methods, improving their effectiveness for DNA clustering and achieving promising results.

Motivated by the limitations of existing DNA clustering tools in balancing runtime scalability and clustering accuracy, we introduce GTree, a tree-structure-based clustering framework optimized for large-scale DNA datasets. The contributions of this paper are threefold.

- Integration of Q-gram, MinHash, and Locality Sensitive Hashing (LSH) into a lightweight pre-screening module that effectively narrows the comparison scope without compromising accuracy, optimizing efficiency in DNA sequence analysis.
- Construction of a prefix tree (trie) for DNA sequence indexing, grouping sequences based on shared prefixes to facilitate rapid retrieval and improve scalability. A majority-label voting strategy is incorporated to mitigate sequencing noise, enhancing clustering stability and robustness.
- Evaluation of GTree on both synthetic and real-world datasets, with comparisons to several baseline methods in terms of clustering accuracy and runtime. Experimental results demonstrate that GTree consistently outperforms existing approaches, particularly in datasets with high sequence similarity or significant sequencing noise.

II. Related Work

In recent years, researchers have proposed various approaches to sequence clustering. A comparison of representative DNA clustering methods is provided in Table I.

A. Edit Distance-based Clustering Approaches

Traditional methods for DNA clustering based on edit distance have shown strong performance in ensuring high clustering accuracy. For example, Starcode [15] introduces a trie tree search mechanism, which is particularly effective for short sequence scenarios, offering both high accuracy and good applicability. Another notable approach, MeShClust [13], incorporates the mean drift algorithm into the DNA clustering process, thereby broadening the technique's applicability across various sequence lengths. Additionally, Rashtchian *et al.* [18] present a distributed approximate clustering framework that combines local computation with communication optimization, significantly enhancing the scalability of the clustering process.

B. Hashing-based Clustering Approaches

In order to reduce the comparison overhead, several methods incorporate hash structures to facilitate more efficient clustering. For instance, DUHI [16] constructs a dynamically updated core index set, which, when combined with hash-based search, enables efficient clustering. GradHC [17] introduces a progressive hash clustering framework that integrates the Q-gram feature, MinHash signature, and LSH filtering mechanism. This approach optimizes both accuracy and scalability, particularly in scenarios involving long chains and small clusters, demonstrating excellent robustness.

C. Machine Learning-based Clustering Approaches

In recent years, some studies have explored the use of neural networks and learning-based representations for DNA sequence clustering. For example, ADRS [14] employs a multilayer perceptron to extract features from DNA sequences and adaptively selects a dimensionality reduction strategy before applying k-means for final clustering. Abraham *et al.* [19] skip the traditional base recognition process and directly cluster nanopore sequencing signals, significantly reducing processing time and enhancing clustering effectiveness.

D. Tree-structured Clustering Approaches

Tree-structured clustering approaches leverage hierarchical or tree-based data structures to efficiently organize and cluster sequences. These methods are particularly effective in managing large-scale datasets by reducing the complexity of sequence comparisons. One such approach is Clover [8], which introduces a tree-structured, indexed linear-time clustering algorithm that enables efficient clustering without the need for global comparisons. By constructing a quad-tree retrieval structure for the front, middle, and back segments of the core sequences, and incorporating a node drift tolerance mechanism, Clover significantly improves memory usage and clustering speed. This method stands out as one of the few structural clustering algorithms that remains stable even when handling datasets on the scale of tens of millions of sequences.

III. Materials and Methods

In this section, we present the design and implementation of GTree, a scalable and accurate clustering algorithm specifically tailored for noisy DNA reads. GTree builds on the core principles of Clover [8], which utilizes a tree-based structure to incrementally cluster sequences.

To further enhance both efficiency and robustness, GTree introduces two key extensions: 1) a MinHash-based candidate pre-filtering module that significantly reduces the scope of comparisons before matching, and 2) a label voting mechanism to improve cluster label consistency after the core clustering process. Unlike traditional methods, which rely on exhaustive pairwise comparisons, GTree first employs Q-gram and MinHash features for candidate sequence filtering via LSH. It then performs drift-tolerant structure matching inspired by Clover's indexed tree alignment. Finally, a simple but effective label consolidation step is applied to further refine clustering accuracy.

TABLE I
COMPARISON OF REPRESENTATIVE DNA SEQUENCE CLUSTERING METHODS

Method	Technical Core	Advantages	Limitations
Starcode [15]	Edit distance, Trie search	High precision on short sequences	Poor scalability to large datasets
DUHI [16]	Hash indexing, Core updating	Fast clustering, Supports dynamic updates	Accuracy sensitive to redundancy
GradHC [17]	Q-gram, MinHash, LSH	High precision, Robust in diverse scenarios	Signature generation is time-consuming
Clover [8]	Tree-based clustering with node shifting	Linear time, Low memory usage	Lacks pre-filtering, Redundant comparisons
ADRS [14]	MLP-based feature learning, K-means	Learns optimal feature embeddings	Accuracy depends on training data

A. MinHash-based Candidate Pre-filtering

To accelerate the structure comparison phase, we introduce a lightweight pre-filtering mechanism that efficiently narrows the set of candidate core sequences for each incoming read. This mechanism integrates q-gram-based sequence representation, MinHash signature construction, and LSH retrieval, serving as a key component for improving runtime scalability.

Let the DNA alphabet be $\Sigma = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$. Given a sequence $x \in \Sigma^L$ and a window length $q \geq 1$, we represent x as the set of its contiguous substrings of length q:

$$\mathcal{S}_q(x) \;=\; \big\{\, x_i x_{i+1} \cdots x_{i+q-1} \;\big|\; 1 \leq i \leq L-q+1 \,\big\}. \quad (1)$$

We adopt *set semantics* when constructing MinHash, meaning that duplicate q-grams do not alter $\mathcal{S}_q(x)$. If weighted occurrences are required, techniques such as consistent weighted MinHash can be applied, though this is beyond the scope of the present work.

For a set $S \subseteq \Sigma^q$, let k independent hash functions $h_1, \ldots, h_k : \Sigma^q \to \mathbb{N}$ be given. The MinHash signature $m(S) \in \mathbb{N}^k$ is then defined as:

$$m(S) = \big(m_1(S), \ldots, m_k(S)\big), \;\; m_i(S) = \min_{g \in S} h_i(g). \quad (2)$$

For two sets $A, B \subseteq \Sigma^q$, the *Jaccard similarity* is defined as:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}. \quad (3)$$

By the MinHash property, the probability that the i-th signature components of A and B are equal is exactly their Jaccard similarity, i.e., $\Pr[m_i(A) = m_i(B)] = J(A,B)$. Using k independent hashes, we obtain the unbiased estimator as follows:

$$\widehat{J}(A,B) = \frac{1}{k} \sum_{i=1}^{k} \mathbf{1}\{m_i(A) = m_i(B)\}, \quad (4)$$

where $\mathbf{1}\{\cdot\}$ is the indicator function.

B. Banding-Based LSH Indexing and Retrieval

Let $k = br$ with integers b (the number of bands) and r (the number of rows per band). We partition the MinHash signature $m(S)$ into b bands, each consisting of a contiguous block of r components:

$$\mathbf{m}^{(j)}(S) \;=\; \big(m_{(j-1)r+1}(S), \ldots, m_{jr}(S)\big), \; j = 1, \ldots, b. \quad (5)$$

Each band is hashed into a *bucket*. Two signatures that are identical in at least one band collide in a bucket and are therefore considered candidates. If two sets have Jaccard similarity s, the probability that they collide in at least one band is given by:

$$P_{\text{cand}}(s) \;=\; 1 - \big(1 - s^r\big)^b. \quad (6)$$

At query time, the MinHash signature of the read is computed, its b band buckets are retrieved, and the union of these buckets forms the candidate set. Structure comparison is then carried out only on this reduced set of candidates (optionally further limited to the Top-K by a lightweight signature-level similarity score).

1) An Example: Consider three core sequences with $q = 3$:

- r_1: $\mathtt{ACTGAC} \to \{\mathtt{ACT}, \mathtt{CTG}, \mathtt{TGA}, \mathtt{GAC}\}$
- r_2: $\mathtt{GATCGA} \to \{\mathtt{GAT}, \mathtt{ATC}, \mathtt{TCG}, \mathtt{CGA}\}$
- r_3: $\mathtt{ACTGAT} \to \{\mathtt{ACT}, \mathtt{CTG}, \mathtt{TGA}, \mathtt{GAT}\}$

Let $k = 3$ hash functions (h_1, h_2, h_3) be applied to the eight distinct q-grams introduced above. As illustrated in Table II, we assign fixed hash values so that the resulting MinHash signatures are reproducible.

TABLE II
THE SETTINGS OF HASH VALUES

q-gram	h_1	h_2	h_3
ACT	18	21	10
CTG	**15**	**9**	8
TGA	23	14	5
GAC	17	18	**3**
GAT	25	11	**2**
ATC	**22**	12	7
TCG	30	**8**	6
CGA	28	19	**1**

From the definition $m_i(S) = \min_{g \in S} h_i(g)$, we obtain:

$$\text{MinHash}(r_1) = [15,\ 9,\ 3],$$
$$\text{MinHash}(r_2) = [22,\ 8,\ 1],$$
$$\text{MinHash}(r_3) = [15,\ 9,\ 2].$$

Now consider a new read $\mathtt{CTGATC}$ with q-grams $\{\mathtt{CTG}, \mathtt{TGA}, \mathtt{GAT}, \mathtt{ATC}\}$. Its signature is

$$\text{MinHash}(\text{read}) = [15,\ 9,\ 2]. \quad (7)$$

LSH retrieval. Take $b = 3$ bands and $r = 1$ row per band (so $k = br = 3$). The three bands of the read are $[15], [9], [2]$. We insert/query by band equality:

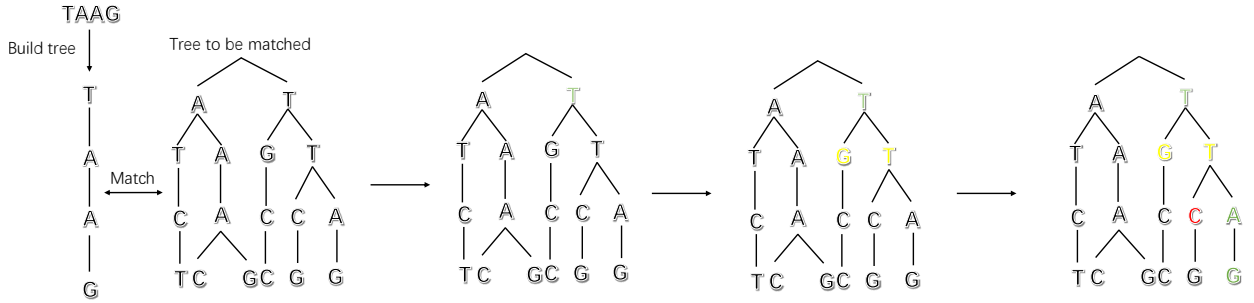

Fig. 2. Horizontal drift: For the sequence [TAAG], we first construct a tree structure. The first node matches, but no nodes match at the second level. We can shift it to either a T-node or a G-node, with the condition that the two consecutive nodes after the drifted node cannot drift again. This rules out the branches TGCC and TTCG. The final match is TTAG, with a drift count of 1 [8].

- Band 1 (15) collides with cores whose first component is 15: $\{r_1, r_3\}$.
- Band 2 (9) collides with cores whose second component is 9: $\{r_1, r_3\}$.
- Band 3 (2) collides with cores whose third component is 2: $\{r_3\}$.

The union yields the candidate set $\{r_1, r_3\}$. A cheap signature-level score (e.g., number of coordinate matches) ranks r_3 first $(3/3$ equal) and r_1 second $(2/3$ equal). Structure comparison is then performed only on these candidates (or the Top-K of them), instead of scanning the entire core set.

Discussion This design respects the correct pipeline order: *sequence* $\rightarrow$ *q-gram set* $\rightarrow$ *MinHash signature* $\rightarrow$ *LSH buckets* $\rightarrow$ *candidate cores* $\rightarrow$ *structure comparison*. In practice, it reduces the average number of structure comparisons from hundreds to a small constant (e.g., $K = 20$), while adding only minimal overhead because both MinHash computation and LSH lookup are highly efficient.

C. Structure-Based Matching with Drift Tolerance

After candidate core sequences are selected by the MinHash pre-filtering module, GTree performs a structure-based matching step to assign the input read to the most appropriate cluster. This module is adapted from the core indexing and retrieval mechanism of the Clover algorithm [8], which enables efficient approximate matching without requiring full-sequence alignment. Each core sequence in GTree is partitioned into three segments, namely, prefix, middle, and suffix, corresponding to the beginning, center, and end of the DNA strand. For each segment, a separate 4-ary prefix tree (trie) is constructed over the DNA alphabet A, C, G, T, yielding a lightweight and error-tolerant indexing structure. During the matching phase, an incoming read is segmented in the same way and traversed through the corresponding tries of its candidate cores.

To tolerate sequencing errors and local shifts, GTree inherits Clover's concept of drift-tolerant matching. Specifically:

- **Horizontal drift** allows a small number of base mismatches (e.g., substitutions) within each segment, enabling tolerance against point mutations.
- **Vertical drift** accommodates insertions or deletions by permitting limited positional shifts in the traversal path, correcting for short indels or misalignments.

Fig. 2 and Fig. 3 illustrate intuitive examples of horizontal drift and vertical drift, respectively, in the tree-structured retrieval process. A read is considered successfully matched if all three of its segments align with the corresponding segments of a core sequence within the allowed drift thresholds. If this condition is not met, the read is promoted to a new core and added to the index structure. This structure-based matching strategy is both efficient and robust. By avoiding costly edit distance computations, it maintains the flexibility to tolerate sequencing errors. Furthermore, because matching is restricted to the small set of candidates identified by the pre-filtering stage, GTree achieves near-constant matching time, independent of the overall size of the core set.

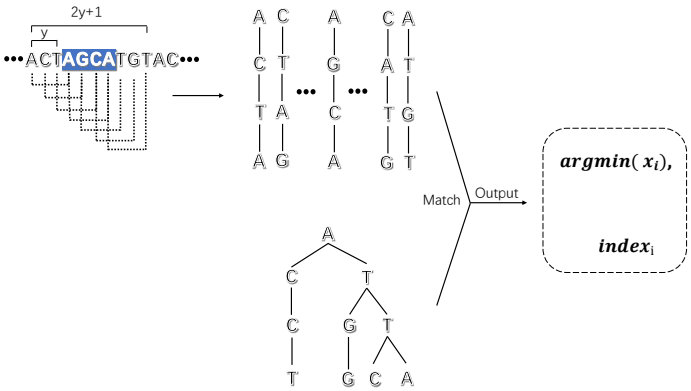

Fig. 3. Vertical drift: When selecting an interval, we identify the one with the maximum vertical displacement before and after the original interval (where $y = 3$). This enables the construction of multiple trees within a larger interval of size $2y + 1$, which are then matched with the target tree. The minimum horizontal drift value and corresponding indicator are then output [8].

D. Majority Label Voting for Cluster Output Consistency

To further enhance the reliability of clustering results, GTree incorporates a lightweight post-processing strategy based on majority label voting. This module aims to improve the consistency between predicted cluster assignments and ground-truth labels, particularly under noisy conditions such as sequencing errors, primer residues, or ambiguous barcode regions.

Once the read set is fully clustered, GTree iterates over each cluster to analyze the distribution of labels associated with its member sequences. Let $C = \{s_1, s_2, \ldots, s_n\}$ represent a cluster with member sequences s_i, each having a reference label l_i. The dominant label l^* is defined as:

$$l^* = \arg\max_{l} \sum_{i=1}^{n} \mathbf{1}(l_i = l). \tag{8}$$

The final label for the cluster is set to l^*, which corresponds to the most frequent label within the cluster. Optionally, reads whose labels do not match l^* can be flagged as outliers or excluded during accuracy evaluation, helping to minimize the impact of misassigned reads near the cluster boundaries.

In contrast to more complex correction strategies, such as reclustering or reverse assignment (e.g., used in GradHC [17]), this voting-based mechanism incurs negligible computational cost while preserving the original clustering structure. It offers an effective and practical trade-off between robustness and efficiency.

E. Algorithm Description

GTree operates within an incremental clustering framework, maintaining a dynamic set of core sequences that guide the classification of incoming reads. Initially, the core set is empty. As each read is processed, it is either assigned to an existing cluster or promoted to a new core sequence based on a two-stage evaluation: fast candidate pre-filtering and structure-based matching. Each core sequence is indexed using a three-part prefix tree structure, consisting of front, middle, and back segments. This structure facilitates efficient approximate retrieval, even in the presence of sequencing errors. Upon receiving a new read, GTree begins by performing feature-based candidate pre-filtering. Specifically, the read's Q-gram profile is converted into a MinHash signature, which is then queried against an LSH index. This process retrieves a top-K subset of similar core candidates, significantly reducing the number of structural comparisons required.

For each retrieved candidate, GTree performs drift-tolerant structure-based matching. The algorithm checks whether the read aligns with all three segments of the prefix tree, ensuring that the allowed thresholds for base substitutions (horizontal drift) and position shifts (vertical drift) are not exceeded. If a match is found, the read is assigned to the corresponding cluster. Otherwise, the read is designated as a new core sequence and added to the index. This process is repeated for all input reads. Since the candidate set size is bounded (typically $K \ll |\text{core set}|$) and tree retrieval operates in constant time per segment, the per-read time complexity remains effectively sub-linear with respect to the growing core set size. Additionally, by avoiding global alignment, GTree allows for fast and memory-efficient clustering of large-scale, noisy datasets.

At the end of the clustering phase, GTree applies a lightweight post-processing step to enhance label consistency. For each cluster, the dominant label is determined via majority voting and assigned as the final cluster label. Optionally, outlier reads with inconsistent labels can be excluded to further enhance robustness.

The complete clustering workflow of GTree is summarized in Algorithm 1, which combines feature hashing, structural indexing, and statistical voting in a tightly integrated design.

Algorithm 1 GTree: Sequence Clustering with MinHash Pre-filtering

Input: DNA read set $S = \{s_1, s_2, \ldots, s_n\}$; clustering parameters P
Output: Clustered result set $\mathcal{C}$; core sequence set $\mathcal{R}$

Initialize core set $\mathcal{R} \leftarrow \emptyset$ and cluster set $\mathcal{C} \leftarrow \emptyset$
Compute Q-gram features and MinHash signatures for all reads in S
Build LSH index from initial core signatures
for each read $s_i \in S$ **do**
  Extract Q-gram features and MinHash signature of s_i
  Retrieve top-K candidate cores $\mathcal{R}_{\text{cand}}$ via LSH
  `matched` $\leftarrow$ `False`
  for each $r_j \in \mathcal{R}_{\text{cand}}$ **do**
    Perform drift-tolerant structural matching with r_j
    if successful match **then**
      Assign s_i to cluster $\mathcal{C}_{r_j}$
      `matched` $\leftarrow$ `True`; **break**
    end if
  end for
  if `matched` is `False` **then**
    Add s_i to $\mathcal{R}$; update index tree and LSH
    Initialize new cluster $\mathcal{C}_{s_i} \leftarrow \{s_i\}$
  end if
end for
return Cluster set $\mathcal{C}$ and core set $\mathcal{R}$

F. Complexity Analysis

Let n be the number of reads, m the number of current core sequences, K the number of candidates returned by the MinHash+LSH filter, and d the maximum depth of the index tree. The parameter k denotes the dimension of the MinHash signature.

1) Time Complexity: For time complexity, both Clover and GTree process each read once, giving an overall complexity of $\mathcal{O}(n)$. In Clover, each new read is compared against all m cores, causing the per-read cost to grow with m as the dataset expands. In contrast, GTree uses the MinHash+LSH pre-filter to retrieve only K candidate cores for structure-based matching, making the per-read cost effectively constant and independent of m. While the asymptotic complexity remains the same, GTree achieves a substantially smaller constant factor, leading to faster practical runtime.

2) Space Complexity: For space complexity, both Clover and GTree maintain a multi-tree index for core sequences, requiring $\mathcal{O}(m \cdot d)$ memory. In addition, GTree stores a k-dimensional MinHash signature for each read and an LSH index, which incurs an extra $\mathcal{O}(n \cdot k)$ space. This overhead grows linearly with n and remains manageable in practice.

Overall, GTree preserves the asymptotic complexity of Clover while enhancing scalability through fixed-size candidate pre-filtering, resulting in lower runtime on large datasets.

IV. Experiments

We conduct comprehensive experiments on both synthetic and real-world DNA storage datasets to evaluate the performance of GTree.

TABLE III
DETAILS OF EXPERIMENTAL DATASETS

Dataset	Reads	Read Length	Original Sequences	Data Source	Generation Method
I18-S3-R1-001	15,169,628	60 bp	16,383	[10]	Real
ERR1816980	14,654,644	152 bp	72,000	[5]	Real
P10-5-BDDP210000009	16,217,014	150 bp	209,283	[20]	Real
Simulated (10M)	10,000,000	160 bp	40,000	—	Random generation
Simulated (1M)	1,000,000	160 bp	5,000	—	Random generation
Simulated (0.1M)	100,000	160 bp	1,000	—	Random generation
Simulated (0.01M)	10,000	160 bp	500	—	Random generation

A. Experimental Setup

1) Datasets: We used a total of three existing open-source real datasets, along with four simulated datasets of varying scales, in our experiments. The source information for the experimental data is summarized below.

- **ERR1816980.** Erlich *et al.* [5] pioneered a DNA data storage architecture using digital fountain codes, successfully creating a storage unit with 152 ultra-long strands (72,000 base pairs). The experiment's original sequencing data, ERR1816980, serves as a benchmark test set, validated by the Illumina Miseq V4 sequencing system across multiple dimensions.
- **P10-5-BDDP210000009.** Song *et al.* [20] developed a directed coding architecture, creating a storage matrix with 210,000 directionally coded DNA strands and achieving a 6 MB data density per batch. The original sequencing data of P10-5-BDDP210000009 is chosen as the core validation sample.
- **I18-S3-R1-001.** Antkowiak *et al.* [10] introduced a DNA storage system based on large-scale parallel photolithography, using mask-free photolithography to synthesize 16,383 sequences of length 60. Their experiments revealed error rates of 2.6% for substitutions, 6.2% for deletions, and 5.7% for insertions. The sequencing file I18-S3-R1-001 was selected as the experimental dataset for validation.
- **Synthetic datasets.** We create a "reference template library" by generating random raw sequences, which are then perturbed to introduce sequencing errors (insertions, deletions, substitutions) with a controllable error rate. The dataset includes a sequence index and true label file for clustering and accuracy analysis. The tool supports flexible adjustments to sequence count, length, error patterns, and copies, enabling comprehensive scalability testing.

A detailed description of these datasets, including the characteristics of the real data, is provided in Table III.

2) Data Preprocessing: In DNA data storage, real sequencing datasets (e.g., P10-5-BDDP210000009 and ERR1816980) typically consist of paired-end reads. Each DNA fragment is sequenced from both directions, generating two FASTQ files (R1 and R2). Since individual reads are often shorter than the full DNA strand, merging R1 and R2 into complete sequences is essential before clustering.

We first use Paired-End reAd mergeR (PEAR) to merge the paired reads. PEAR is a fast and memory-efficient tool optimized for Illumina paired-end data. It automatically detects overlapping regions between R1 and R2 and stitches them into high-quality single-end reads, enhancing alignment accuracy and removing primer regions that could interfere with clustering. Next, Bowtie2 is applied to align the merged reads to the corresponding reference sequences. The alignment is performed in default mode, which is effective for datasets with low error rates, such as ERR1816980 and P10. The output is a SAM file containing the alignment results. We then parse the SAM file to extract the matched reference ID (i.e., Packet ID) for each successfully aligned read. Each read and its corresponding reference label are recorded in a text file with the format: `Label Sequence`, where each line represents a DNA read and its assigned cluster label. Unaligned reads are excluded from the final labeled dataset to ensure consistency and labeling reliability. Finally, to eliminate interference from experimental artifacts, sequencing primers are removed during preprocessing. This ensures that clustering is based solely on the core informative regions of the reads, consistent with the internal logic of Clover and similar tools. By combining PEAR and Bowtie2, we construct robust labeled datasets from raw sequencing reads, enabling accurate evaluation of clustering algorithms under realistic conditions.

3) Evaluation Metrics: To systematically evaluate the performance of GTree and Clover across various datasets, we define two core evaluation metrics:

- **Runtime Efficiency:** The total time (in seconds) required to complete the clustering process. This metric measures the efficiency of each algorithm in handling large-scale DNA data.
- **Clustering Accuracy:** The proportion of reads that are correctly assigned to their original reference-derived clusters. Ground truth labels are generated by aligning reads to reference sequences using Bowtie2, and clustering results are compared accordingly.

TABLE IV
CLUSTERING PARAMETERS FOR CLOVER AND GTREE

Dataset	Count	Depth	H. Shift	V. Shift
I18-S3-R1-001	4	20	1	3
ERR1816980	4	15	3	3
P10-BDDP	4	20	3	3
Synthetic	4	15	3	3

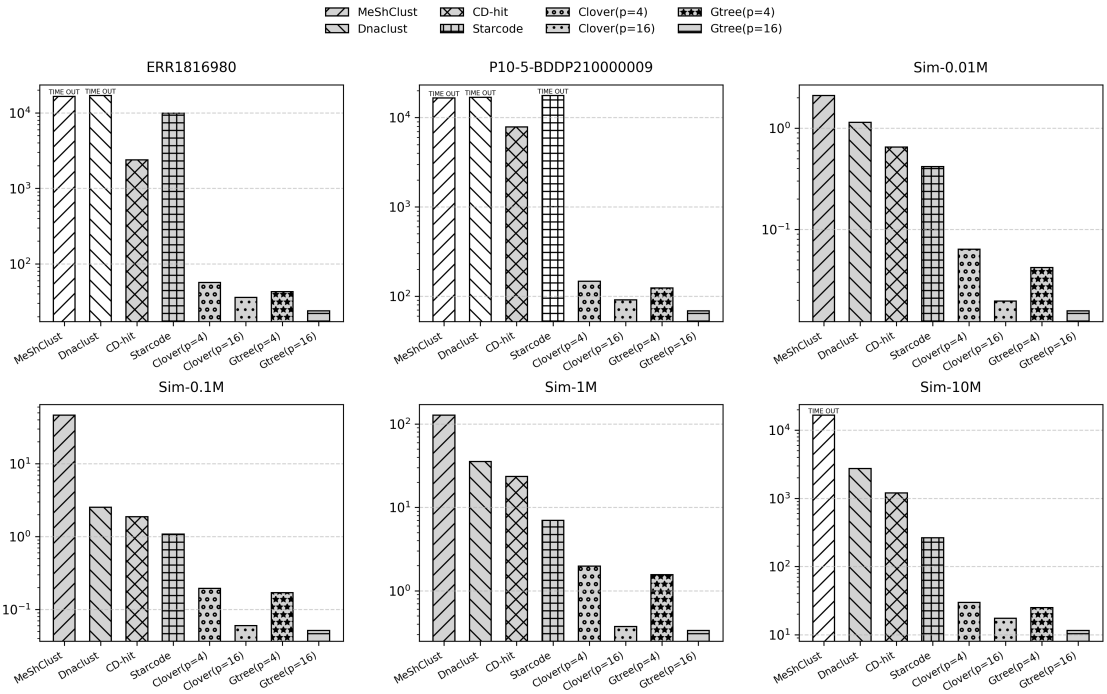

Fig. 4. Runtime comparison on all datasets. "Timeout" indicates the algorithm did not complete within the allocated time limit.

4) Baselines: GTree is compared against several representative baselines, including alignment-based, hashing-based, and learning-based clustering methods.

- **CD-HIT** [21]: A greedy local-alignment-based method originally designed for protein and DNA clustering.
- **MeShClust** [13]: A machine learning-based alignment-free tool for DNA clustering.
- **Starcode** [15]: A trie-based clustering algorithm using edit distance thresholding.
- **DNACLUST** [22]: A fast, heuristic-based DNA clustering tool that uses k-mer similarity and greedy aggregation to efficiently group sequences without full alignments.
- **Clover** [8]: A tree-structured clustering method using node-shift-tolerant alignment without pre-filtering.

All methods are executed with default or recommended parameters. For learning-based methods like ADRS [14], pre-trained models are used when available; otherwise, they are excluded from real-data comparisons due to generalization concerns. To ensure fairness, Clover and GTree use consistent parameter settings, as outlined in Table IV. Parameters such as the number of prefix trees, tree depth, and node shift are tuned based on dataset characteristics. Typically, a tree depth between 15 and 20 strikes a balance between clustering precision and memory usage, while tree counts are configured to cover both sequence ends and middle regions.

B. Results and Analysis

1) Runtime Comparison: Fig. 4 presents the runtime results across all datasets. GTree consistently achieves the lowest runtime among all tested methods, outperforming Clover by approximately 20% on average, while maintaining similar or higher accuracy. In large-scale datasets (e.g., 10M synthetic reads), several baseline methods (MeShClust and DNACLUST) fail to complete within the time limit, whereas GTree maintains stable performance. Starcode shows competitive speed on small datasets but suffers from accuracy degradation (see Table VI). The improvement of GTree over Clover primarily stems from the MinHash+LSH pre-filtering and optimized trie-based core matching, which reduces unnecessary comparisons without compromising accuracy.

2) Accuracy Comparison: Table V presents the clustering accuracy of GTree and Clover across six datasets, covering both real-world and synthetic scenarios. Overall, GTree maintains very high accuracy on all datasets, consistently exceeding 99.8% on real datasets such as ERR1816980 and P10-5-BDDP210000009. Even on the large-scale 10M synthetic dataset, where high error rates and massive sequence counts make clustering more challenging, GTree reaches 99.96% accuracy, slightly surpassing Clover. Compared with Clover, GTree achieves either comparable or higher accuracy on every dataset, with notable improvements on ERR1816980 and the 10M synthetic set. This improvement can be attributed to GTree's enhanced pre-filtering and trie-based candidate grouping, which help preserve global sequence similarity while reducing mismatches. On datasets with moderate error rates (e.g., 0.01M–1M synthetic sets), both methods achieve near-perfect accuracy, demonstrating that GTree's structural filtering does not compromise clustering precision.

For the smaller I18-S3-R1-001 dataset (Table VI), GTree delivers competitive accuracy with significantly faster runtime than most baselines. While Starcode matches Clover in runtime, its accuracy is much lower (23.17%), indicating poor clustering quality. Both GTree and CD-HIT maintain over 99.9% accuracy despite the high base error rate, but GTree outperforms in clustering speed on the same hardware. These results confirm that GTree retains Clover's strong clustering performance while consistently improving both accuracy and efficiency.

TABLE V
CLUSTERING ACCURACY COMPARISON BETWEEN CLOVER AND GTREE

Dataset	**Clover** ($p = 4$)	**Clover** ($p = 16$)	**GTree** ($p = 4$)	**GTree** ($p = 16$)
ERR1816980	99.43%	99.63%	99.81%	99.92%
P10-5-BDDP210000009	99.79%	99.79%	99.83%	99.81%
0.01M	99.81%	100.00%	100.00%	100.00%
0.1M	100.00%	100.00%	100.00%	100.00%
1M	99.93%	99.97%	99.99%	99.99%
10M	99.37%	99.67%	99.90%	99.96%

TABLE VI
RUNTIME AND ACCURACY COMPARISON ON I18-S3-R1-001 DATASET

Metrics	**GTree (ours)**	**Clover**	**UCLUST**	**CD-HIT**	**DNACLUST**	**Starcode**	**MeShClust**
Runtime (s)	4.12	5.2	80.1	144.8	648.4	10.23	7947.22
Accuracy (%)	99.99	99.94	98.53	99.96	85.99	23.17	–

V. CONCLUSION

In this paper, we propose GTree, a tree-structured clustering algorithm designed to enhance both the scalability and accuracy of DNA sequence clustering in large-scale DNA storage systems. GTree integrates a prefix trie structure with node-shift-tolerant alignment, enabling efficient grouping of noisy reads without relying on costly global alignments or external filtering. Compared with existing methods such as Clover, Starcode, and CD-HIT, GTree demonstrates superior clustering accuracy, reduced runtime, and lower memory usage across both synthetic and real-world datasets. Our extensive experiments show that GTree achieves over 99.4% clustering accuracy on multiple benchmarks while being up to 30% faster than Clover. The trie-based architecture not only accelerates alignment but also enhances robustness to sequencing errors, making GTree highly suitable for practical deployment in large-scale DNA storage.

ACKNOWLEDGMENT

This work was supported by the National Key Research and Development Program of China (2025YFC3409900 and 2020YFA0712100) and the Emerging Frontiers Cultivation Program of Tianjin University Interdisciplinary Center. Huaming Wu is the corresponding author.

REFERENCES

[1] Y. Dong, F. Sun, Z. Ping, Q. Ouyang, and L. Qian, "Dna storage: research landscape and future prospects," *National Science Review*, vol. 7, no. 6, pp. 1092–1107, 2020.

[2] D. Rydning, J. Reinsel, and J. Gantz, "The digitization of the world from edge to core," *Framingham: International Data Corporation*, vol. 16, pp. 1–28, 2018.

[3] N. Goldman, P. Bertone, S. Chen, C. Dessimoz, E. M. LeProust, B. Sipos, and E. Birney, "Towards practical, high-capacity, low-maintenance information storage in synthesized dna," *Nature*, vol. 494, no. 7435, pp. 77–80, 2013.

[4] G. M. Church, Y. Gao, and S. Kosuri, "Next-generation digital information storage in dna," *Science*, vol. 337, no. 6102, pp. 1628–1628, 2012.

[5] Y. Erlich and D. Zielinski, "Dna fountain enables a robust and efficient storage architecture," *Science*, vol. 355, no. 6328, pp. 950–954, 2017.

[6] L. Organick, S. D. Ang, Y.-J. Chen, R. Lopez, S. Yekhanin, K. Makarychev, M. Z. Racz, G. Kamath, P. Gopalan, B. Nguyen *et al.*, "Random access in large-scale dna data storage," *Nature Biotechnology*, vol. 36, no. 3, pp. 242–248, 2018.

[7] L. Ceze, J. Nivala, and K. Strauss, "Molecular digital data storage using dna," *Nature Reviews Genetics*, vol. 20, no. 8, pp. 456–466, 2019.

[8] G. Qu, Z. Yan, and H. Wu, "Clover: tree structure-based efficient dna clustering for dna-based data storage," *Briefings in Bioinformatics*, vol. 23, no. 5, p. bbac336, 2022.

[9] J. Jeong, S.-J. Park, J.-W. Kim, J.-S. No, H. H. Jeon, J. W. Lee, A. No, S. Kim, and H. Park, "Cooperative sequence clustering and decoding for dna storage system with fountain codes," *Bioinformatics*, vol. 37, no. 19, pp. 3136–3143, 2021.

[10] P. L. Antkowiak, J. Lietard, M. Z. Darestani, M. M. Somoza, W. J. Stark, R. Heckel, and R. N. Grass, "Low cost dna data storage using photolithographic synthesis and advanced information reconstruction and error correction," *Nature Communications*, vol. 11, no. 1, p. 5345, 2020.

[11] J. A. Hartigan and M. A. Wong, "Algorithm as 136: A k-means clustering algorithm," *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, vol. 28, no. 1, pp. 100–108, 1979.

[12] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, ser. KDD'96, 1996, p. 226–231.

[13] B. T. James, B. B. Luczak, and H. Z. Girgis, "Meshclust: an intelligent tool for clustering dna sequences," *Nucleic Acids Research*, vol. 46, no. 14, pp. e83–e83, 2018.

[14] B. Liu and J. Li, "Adrs-cnet: An adaptive dimensionality reduction selection and classification network for dna storage clustering algorithms," *arXiv preprint arXiv:2408.12751*, 2024.

[15] E. Zorita, P. Cusco, and G. J. Filion, "Starcode: sequence clustering based on all-pairs search," *Bioinformatics*, vol. 31, no. 12, pp. 1913–1919, 2015.

[16] P. Wang, B. Cao, T. Ma, B. Wang, Q. Zhang, and P. Zheng, "Duhi: Dynamically updated hash index clustering method for dna storage," *Computers in Biology and Medicine*, vol. 164, p. 107244, 2023.

[17] D. Ben Shabat, A. Hadad, A. Boruchovsky, and E. Yaakobi, "Gradhc: highly reliable gradual hash-based clustering for dna storage systems," *Bioinformatics*, vol. 40, no. 5, p. btae274, 2024.

[18] C. Rashtchian, K. Makarychev, M. Racz, S. Ang, D. Jevdjic, S. Yekhanin, L. Ceze, and K. Strauss, "Clustering billions of reads for dna data storage," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[19] H. Abraham, B. Gahtan, A. Kobovich, O. Leitersdorf, A. M. Bronstein, and E. Yaakobi, "Beyond the alphabet: Deep signal embedding for enhanced dna clustering," *arXiv preprint arXiv:2410.06188*, 2024.

[20] L. Song, F. Geng, Z.-Y. Gong, X. Chen, J. Tang, C. Gong, L. Zhou, R. Xia, M.-Z. Han, J.-Y. Xu *et al.*, "Robust data storage in dna by de bruijn graph-based de novo strand assembly," *Nature communications*, vol. 13, no. 1, p. 5361, 2022.

[21] W. Li and A. Godzik, "Cd-hit: a fast program for clustering and comparing large sets of protein or nucleotide sequences," *Bioinformatics*, vol. 22, no. 13, pp. 1658–1659, 2006.

[22] M. Ghodsi, B. Liu, and M. Pop, "Dnaclust: accurate and efficient clustering of phylogenetic marker genes," *BMC bioinformatics*, vol. 12, no. 1, p. 271, 2011.