Briefings in Bioinformatics, 2022, **23(5)**, 1–16

https://doi.org/10.1093/bib/bbac336

Problem Solving Protocol

Clover: tree structure-based efficient DNA clustering for DNA-based data storage

Guanjin Qu, Zihui Yan and Huaming Wu

Corresponding author: Huaming Wu, Center for Applied Mathematics, Tianjin University, Tianjin, 300072, China. E-mail: whming@tju.edu.cn

Abstract

Deoxyribonucleic acid (DNA)-based data storage is a promising new storage technology which has the advantage of high storage capacity and long storage time compared with traditional storage media. However, the synthesis and sequencing process of DNA can randomly generate many types of errors, which makes it more difficult to cluster DNA sequences to recover DNA information. Currently, the available DNA clustering algorithms are targeted at DNA sequences in the biological domain, which not only cannot adapt to the characteristics of sequences in DNA storage, but also tend to be unacceptably time-consuming for billions of DNA sequences in DNA storage. In this paper, we propose an efficient DNA clustering method termed Clover for DNA storage with linear computational complexity and low memory. Clover avoids the computation of the Levenshtein distance by using a tree structure for interval-specific retrieval. We argue through theoretical proofs that Clover has standard linear computational complexity, low space complexity, etc. Experiments show that our method can cluster 10 million DNA sequences into 50 000 classes in 10 s and meet an accuracy rate of over 99%. Furthermore, we have successfully completed an unprecedented clustering of 10 billion DNA data on a single home computer and the time consumption still satisfies the linear relationship. Clover is freely available at https://github.com/Guanjinqu/Clover.

Keywords: DNA storage, DNA clustering, Tree structure, Levenshtein distance

Introduction

In recent years, along with the influx of new Internet of Things (IoT) devices and the exponential growth in demand for their services, massive amounts of data are being generated and collected [1]. The traditional storage media, e.g. floppy disks, CDs and USB sticks, face daunting challenges and fail to meet the requirements of large capacity and excellent durability [2]. Deoxyribonucleic acid (DNA) has become an attractive medium for long-term information storage due to its advantages of high storage density, capacity and longevity [3–7]. DNA storage technology refers to the use of synthetic DNA consisting of four bases named Adenine (A), Thymine (T), Guanine (G) and Cytosine (C) for digital data storage. With the rapid development of DNA synthesis and sequencing technologies such as single-cell sequencing [8], large-scale DNA storage has become a ground-breaking future-proof storage technology to cope with the explosion of data [9–12].

A complete DNA storage system consists of multiple steps including encoding, DNA synthesis, amplification, sequencing, reading and decoding. One of the dominant difficulties is that synthesizing and sequencing DNA sequences are still far from perfect [13, 14]. As a result, a variety of errors such as base substitutions, deletions, insertions and strand breaks will occur in the original DNA sequence during the processes of DNA synthesis, polymerase chain reaction (PCR) amplification, sequencing and other processes [15, 16]. In addition, the sequenced sequences are disordered and have a large number of repeats, which makes decoding difficult. Clustering is an effective method to improve the success rate of decoding, especially in the case of high error rates or big data [17, 18]. By clustering a large number of sequenced DNA sequences, clustering can reduce duplicate decoding when decoding. Moreover, clustering can be used to fit out candidate sequences by clustering to correct some of the erroneous bases. The DNA storage process and the role of clustering are illustrated in Figure 1.

If we treat DNA clustering as a text clustering problem, then we can also use conventional clustering algorithms for DNA clustering. Since it requires clustering a large number of DNA sequences, the clustering algorithm used must have low computational complexity. However, partition-based clustering algorithms such as K-Means [19] are not suitable for DNA clustering because the number of clusters must be specified in advance and the Euclidean distance must be strictly used as a metric function, despite their low computational complexity. In addition, although density-based clustering algorithms such as DBSCAN [20] are also less computationally complex than quadratic, they require a significant amount of time due to the computation of distance metrics. We will verify that it is nearly impossible to cluster more than 100 000 DNA sequences using the DBSCAN algorithm. To sum up, the traditional clustering algorithms cannot effectively solve the DNA clustering problem.

Currently, there are several DNA clustering methods available, e.g. CD-HIT [21], MeShClust [22], SEED [23], Starcode [24] and UCLUST [25]. CD-HIT [21] is a DNA clustering model that processes sequences from long to short by using a greedy incremental algorithm; however, there is no guarantee that an optimal solution can be found. SEED [23] is an efficient algorithm that uses developed hashing techniques and interval seeding to index sequences. Starcode [24] is one of the effective DNA clustering algorithms

Guanjin Qu is a PhD candidate at the Center for Applied Mathematics, Tianjin University. His research focuses on DNA storage.
Zihui Yan is a PhD candidate at the Center for Applied Mathematics, Tianjin University. Her research focuses on error control coding and communication algorithms.
Huaming Wu is an Associate Professor at the Center for Applied Mathematics, Tianjin University. His research interests include edge computing, internet of things and DNA storage.

Received: May 18, 2022. **Revised:** July 21, 2022. **Accepted:** July 22, 2022

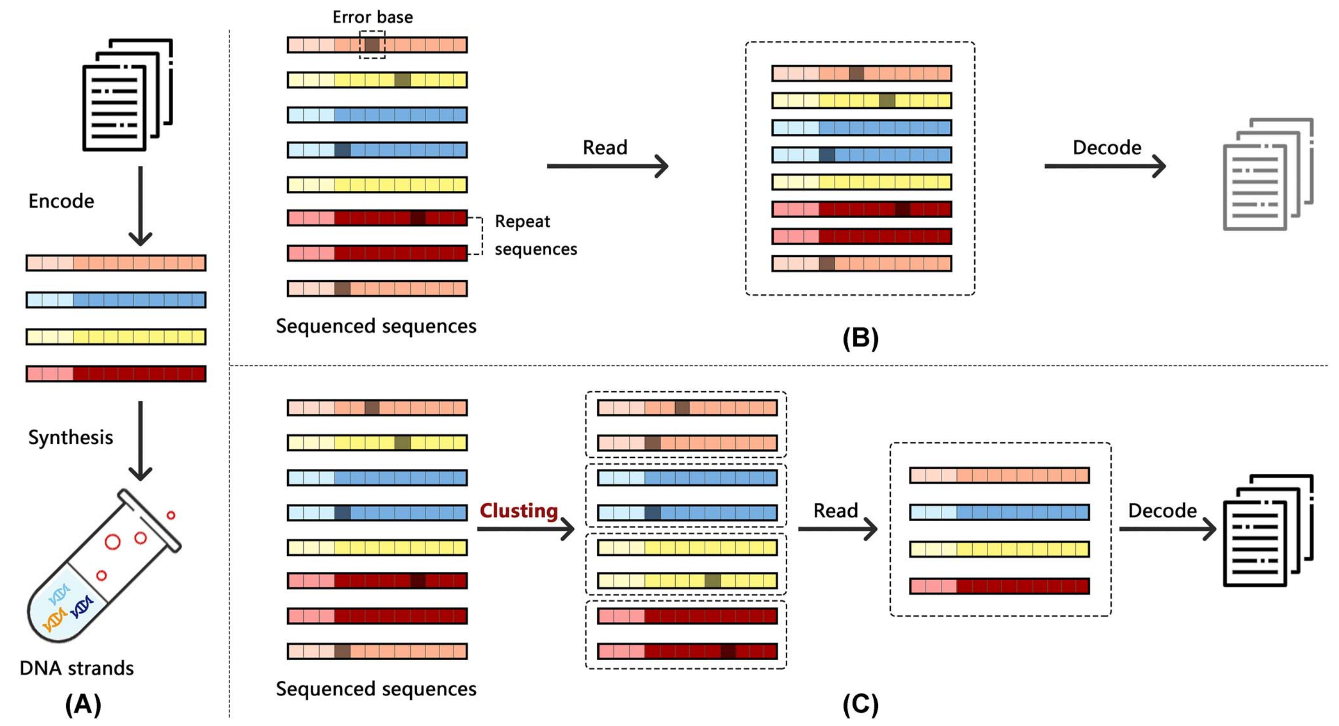

Figure 1. DNA storage process and the role of DNA clustering. **(A)** shows the process of converting a file into a DNA sequence, including encode and synthesis. **(B)** shows the process of decoding DNA storage without clustering, which shows that the sequenced sequence has wrong bases, sequence duplication, etc. It is possible that direct reading and decoding of sequenced sequences may not successfully recover the files. **(C)** shows the decoding process after DNA clustering is used. Clustering can fit homogeneous sequences into a candidate sequence, which in turn improves the decoding efficiency and reduces the error rate. Therefore, using DNA clustering can improve the success rate of decoding.

in open source, which derives the Levenshtein distance mainly by using edit matrices. Meshclust [22] reduces the parameter sensitivity problem by using the mean-shift algorithm. UCLUST [25] is a well-established and widely used clustering method based on USEARCH [25], which clusters sequences by finding common short words between sequences [26]. Unfortunately, the above clustering algorithms are still very time-consuming for the huge amount of data in DNA storage. Antkowiak *et al.* [18] developed a location-sensitive, hash-based clustering algorithm in DNA storage, which reduces the computational cost by using hash tables. Jeong *et al.* [17] proposed a Hamming distance-based clustering method. Additionally, Microsoft has developed a DNA clustering algorithm [27], which uses a minimal hash algorithm to cluster 5 billion pieces of DNA data; however, it still requires reading all the DNA data into memory for clustering, which will consume a lot of memory for storing large-scale DNA data. In general, most of them suffer from greedy algorithms that may not produce optimal clusters and are sensitive to sequence similarity thresholds. Overall, although there are many efficient DNA clustering algorithms available, they still face two difficulties when dealing with the data used in DNA storage, as shown below.

- Existing DNA sequence clustering algorithms focus more on DNA sequences in the biological field, and thus cannot be well adapted to the characteristics of DNA storage data. For instance, sequence similarity is hardly used as a basis for clustering discrimination in the DNA storage field, because DNA sequences are only used as the medium of information in this field, and the differences between their sequences are mainly affected by channel encoding. DNA sequence clustering does not need to strictly consider the similarity between each sequence, as long as the sequences in the clusters are the same original codeword after clustering.
- As the storage capacity of DNA storage field gradually increases, the dataset may contain billions of 200nt oligos under Gigabyte data. The established DNA clustering algorithms rarely have strictly linear computational complexity, and thus often incur unacceptable time consumption for datasets of this order of magnitude.

To address the aforementioned challenges, we propose a novel DNA clustering method termed Clover for DNA storage, in which a multiple tree structure is constructed to search for a specified interval of DNA sequences, thus avoiding the time-consuming computation of the Levenshtein distance. What's more, a node-drift algorithm is developed to counteract the interference caused by errors in the DNA sequence. Compared with several existing DNA clustering algorithms, Clover can cluster a large number of unrecognized DNA sequences, and meanwhile perform error correction and comparison of the clustered DNA sequences, allowing direct output of the original corrected DNA sequences. Therefore, Clover can significantly reduce post-read processing time, and achieve efficient and effective DNA sequence clustering. To sum up, the proposed clustering solution owns the following merits:

- **Low Computational Complexity**: Clover has a standard linear computational complexity, much lower than that of conventional clustering algorithms. We have verified this both through theoretical proofs and experimental simulations.
- **Low Memory Consumption**: Clover allows memory to be automatically released after a DNA sequence has been compared, thus ensuring that memory is not heavily used. The space complexity of our algorithm is only related to the size

of the original sequence and is independent of the number of DNA sequencing sequences, thus preventing the high memory consumption that would result from low computational complexity.

- **High Accuracy**: Clover uses retrieval intervals to classify DNA sequences, which in turn ensures that there are highly overlapping fragments within the same cluster. Theoretically, the probability of misclassification is extremely low. Furthermore, we tested the accuracy of the algorithm on four real-world datasets, including a DNA dataset using photolithographic synthesis with a very high error rate. Experimental results demonstrate that Clover maintains an accuracy rate of over 97% in all cases.
- **Strong Scalability**: Clover is very easy to use and has minimal requirements for computer hardware and systems. We have successfully clustered 10 billion pieces of simulated DNA data on a single desktop computer. As far as we know, no study has ever conducted clustering experiments on DNA data of this order of magnitude. In addition, our algorithm supports parallel computing, and by running it in parallel, the speed of sequence clustering can be greatly improved. Experimental results show that when using distributed computing, Clover can cluster 10 million real DNA sequences to 50 000 categories within 10 s with an accuracy rate of over 99%.

Materials and methods

Clover is a general-purpose DNA sequence clustering algorithm that can quickly cluster a large number of DNA sequences with errors. The clustering process can be briefly described as follows: (i) Clover constructs a core set of sequences, and then compares each of the unclassified sequences with the core set. (ii) If the sequence matches the core set, it is successfully classified into the specified cluster, otherwise, it is added to the core set as a new core sequence. (iii) When comparing the unclassified sequences with the core set, we first construct the core set as an index tree and then compare it with the sequences, thus avoiding the problem of increased comparison time associated with a larger core set. Moreover, we allow node drifting in the tree during retrieval to reduce the impact of sequence errors.

In this section, we will define the tree structure, explain the horizontal drift and vertical drift and describe the overall algorithm.

Tree structure

A schematic diagram of some DNA sequences forming a tree structure is illustrated in Figure 2. Since a DNA sequence can only consist of A, T, G, C, there are at most four nodes under each root, which is why our algorithm is called Clover, because Clover has at most four leaves.

The core of our DNA sequence clustering approach is that it employs a tree structure for DNA sequence clustering. The tree structure is an important nonlinear data structure, in which data elements are organized in branching relationships, much like a tree in nature. To prove the complexity of the algorithm in the following, we first give the definition of the tree structure.

Definition 1.1. A tree is a finite set of $n(n > 0)$ elements, where

(1) Each element is called a node.
(2) There is a particular node, called the root node or root.
(3) Except for the root node, the remaining nodes are divided into a finite number of disjoint sets, where each set is a tree.

We define a tree to have depth L if the tree is constructed from $M(M \geq 0)$ sequences of length L. A retrieval tree structure algorithm is described in Algorithm 1. We next prove that the retrieval time of the tree structure is independent of the number of sequences contained in the tree.

Algorithm 1 Retrieval tree structure algorithm

```
Input: Tree structure and sequence of specified intervals $[t, t+L]$
Output: Result of the retrieval
1: for $i = 1, 2, 3, \cdots, L$ do
2:     Select the $i$-th element in the specified interval $[t, t+L]$ of the sequence
3:     if exists the value of node is equal to that element then
4:         Set that node to root
5:     else
6:         Abort algorithm, output search failure
7:     end if
8:     Output the index value returned when the tree structure has been searched
9: end for
```

Theorem 1.2. *For a tree of depth L consisting of $M(M \geq 0)$ sequences, the computational complexity of retrieving the tree for any sequence is $O(L)$.*

Proof For a specified interval $[t, t + L]$ of any sequence we will retrieve the tree structure according to Algorithm 1. Obviously, no matter how many nodes are included in the tree. For any sequence of L-long intervals, the tree must be retrieved after at most L cycles. Thus, the computational complexity of retrieving the tree structure is $O(L)$. It follows that howsoever many sequences (nodes) the tree contains, it does not affect the time taken to retrieve the tree. ■

Node drift

In order to counteract the interference caused by DNA error bases, we allow some degree of node drift when retrieving the DNA sequence tree. We divide the drift into a horizontal drift and a vertical drift.

Definition 1.3. (*Horizontal drift*): For nodes that do not exist when the tree is retrieved, the rest of the nodes that already exist under the root will be retrieved, and if other nodes exist and the next node can still be matched when the node is a subtree, then drifting to other nodes is allowed.

Definition 1.4. (*Vertical drift*): For a sequence-specific interval $[a, b]$, where $0 \leq a \leq b$, for which the tree should have been retrieved, when the vertical drift is $t(t \leq a)$, then the tree is actually retrieved using the intervals $[a + t, b + t]$ and $[a - t, b - t]$.

Figure 3 shows the node drift. Figure 3A shows the horizontal drift, by which the impact of base substitution errors can be reduced. Figure 3B shows the vertical drift, which reduces the impact of base insertions and deletions in the preamble sequence.

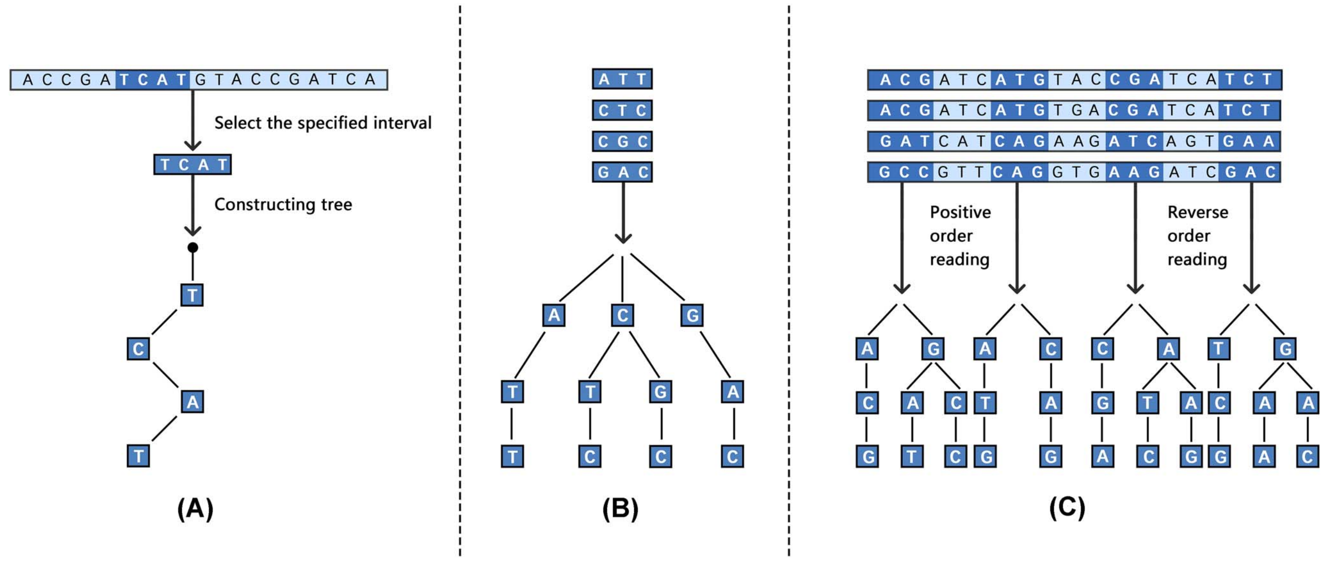

Figure 2. Tree structure diagram. **(A)** represents the first branch of a tree generated by selecting specific intervals [T, C, A, T] in a particular DNA sequence. **(B)** represents the tree structure generated by multiple intervals. **(C)** represents the tree generated by different intervals in multiple DNA sequences, where four sequences belong to each of the three original clusters. The depth of the tree is set to 3, generated by four specified intervals of each sequence: positive base 1 to 3, positive base 7 to 9, reciprocal base 1 to 3 and reciprocal base 7 to 9. The tree structure formed by sequences under the same cluster is identical. This is also the complete way to build a tree structure in Clover.

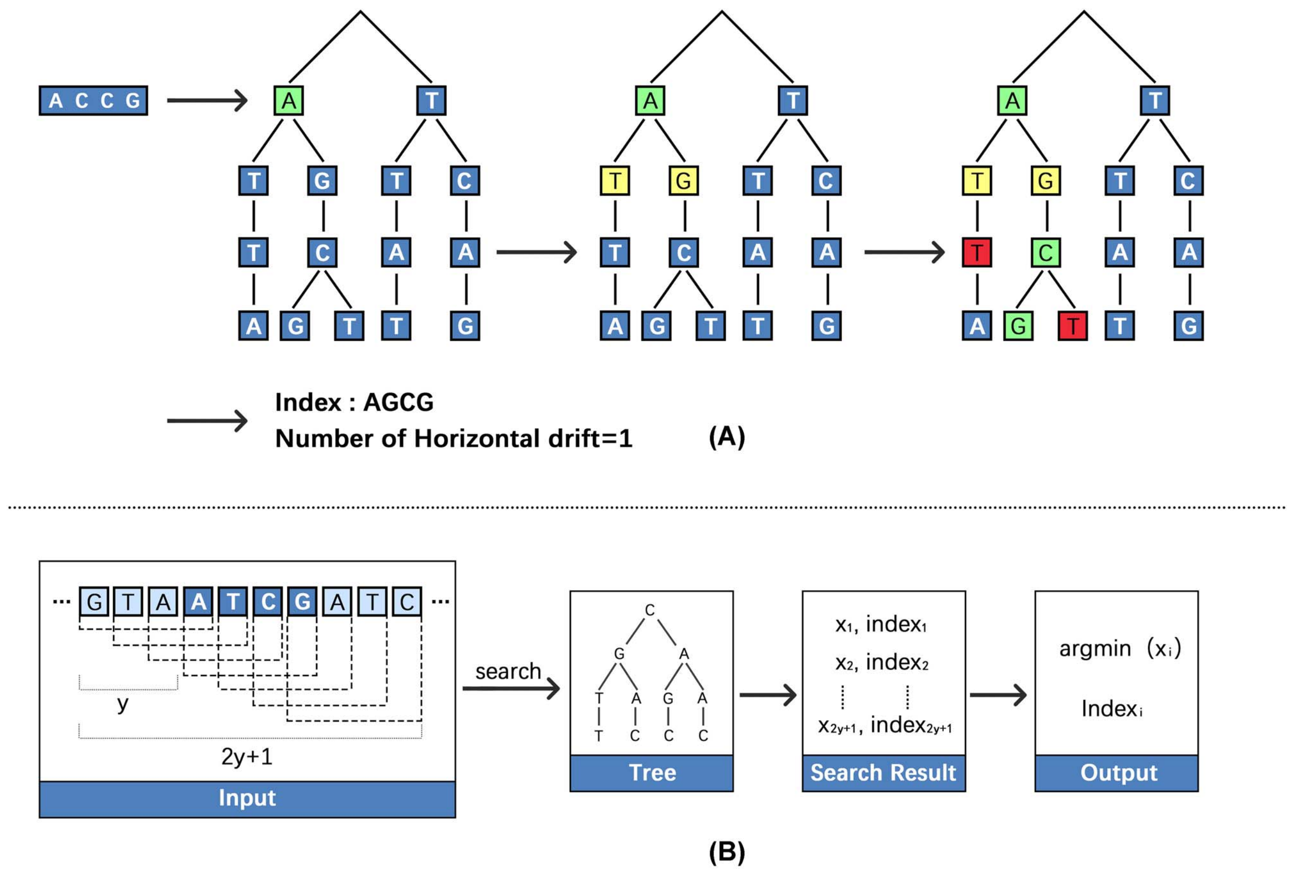

Figure 3. Node drift diagram. **(A)** indicates horizontal drift, for a specific sequence [ACCG], which has no matching node in the second layer of the tree, for which we can candidate drift to T or G nodes, but we require that the subsequent two consecutive nodes of the drift node are not allowed to drift again, so the branches ATTA and AGCT are not satisfied, for which the final match to the branch is AGCG, with a drift count of 1. **(B)** indicates the longitudinal drift, where the longitudinal drift y value is 3. When selecting intervals for tree retrieval, the interval with the most y bits of displacement before and after the original interval is allowed to be selected. After subjecting a total of 2y + 1 intervals to tree retrieval, the algorithm will output the smallest lateral drift value and the corresponding index.

Algorithm description

The Clover algorithmic process is as described in Algorithm 2. Clover contains a core set of sequences. The core set is empty before clustering, but we want the core set to contain the entire original sequences after clustering. We construct multiple tree structures with specified intervals at the front, middle and back ends of the core set. For each sequence that enters the core set, the specified intervals are added to the tree structure. If an unclassified sequence successfully retrieves all the nodes of a sequence in the tree, it is matched to that core sequence in the core set, if not, it is added to the core set as a new sequence and the specified interval is added to the tree structure. The node drift parameter is specified at the time of retrieval so that sequences with errors can still be retrieved successfully.

Due to the nature of the tree structure, the time taken to retrieve the tree structure for the sequences to be tested is not affected, regardless of the expansion of the core set of sequences. The algorithm is executed once for each sequence in the dataset, and the first and last sequences are processed in a theoretically equal amount of time. Therefore, the time complexity of our Clover is linear (even though node drift increases the number of retrievals finitely). In addition, since the algorithm can release memory once the unsorted sequences are read, the space complexity is only related to the depth of the tree and the size of the original sequence, which greatly reduces memory consumption. It should be emphasized that the clustering process does not rely on the global alignment between sequences [28], and we have experimentally verified that the global comparison has little impact on the clustering effect. Moreover, we also allow a one-step global alignment between unclassified sequences and matched core sequences to directly output the error-corrected set of core sequences after clustering, enabling data read simplification. Clover does not contain a global alignment algorithm, and users can embed available mature algorithms. In addition, Clover allows the multi-process mode to increase the execution speed of the algorithm. The dataset will be triaged based on the first end of the sequence, so Clover only allows indices with a process count of four.

Algorithm 2 Clover: DNA sequence clustering algorithm

```
Input: $M$ DNA sequences, lateral drift value $x$
Output: DNA taxonomic cluster $C$, core sequence $R$
 1: Depend on sequence front-end for sequence triage
 2: Initializing the tree structure
 3: for $i = 1, 2, 3, \cdots, M$ do
 4:     Compare the $i$-th sequence with the tree structure
 5:     Generate the closest core sequence $c$ and the number of drifts $n$
 6:     if $n \leq x$ then
 7:         Classify the $i$-th sequence into cluster $c$
 8:         Compare the $i$-th sequence to the core sequence of cluster $c$ globally
 9:     else
10:         Add the $i$-th sequence to the core sequence set and use it as a new cluster
11:         Add the specified interval of the $i$-th sequence to the tree structure
12:     end if
13: end for
```

The overall flow of the Clover DNA sequence clustering algorithm is given in Figure 4. The triaged data will be retrieved with a tree structure that is independent of the tree structure of the different processes. Generally, Clover will build multiple trees for the first, middle and last intervals. In order to improve the efficiency of the algorithm, we will first perform the tree structure for the first and last intervals of the sequence, and any successful tree retrieval (with a lateral drift value generated by the retrieved index below the set threshold) will be considered as a successful match and further retrieval will be stopped. If the retrieval fails, a tree structure retrieval will be performed at the middle end, where a vertical drift will be used to provide a retrieval success rate. If a sequence is eventually successfully retrieved, it will be regarded as a match to an existing cluster. If global alignment is enabled, the sequence will also be globally compared with the matching core sequence and bases with deviations are marked. If a base position is marked too many times, it will be regarded as an error in the core sequence, and then error correction will be performed. If the match is unsuccessful, a new cluster is considered and added to the core sequence set and the tree structure is updated.

Theoretical analysis

We first give the mathematical description of the problem:

Definition 1.5. We assume that the number of sequences is M and they belong to N clusters, respectively, and the original length of the sequences is L. The tree structure has n and length t. We set the horizontal drift to x and the vertical drift to y. We assume that the three error rates for delete, insertion and substitution are p_s, p_d and p_i, respectively. We assume that the true cluster of the sequence is C, where $C = \{c_1, c_2, \cdots, c_J\}$.

Next, to evaluate the effect of clustering, we give the definition of clustering accuracy as follows:

Definition 1.6. (*Accuracy*): We define $\widetilde{C}$ to be the clustering result output by Clover, where $\widetilde{C} = \{\widetilde{c}_1, \widetilde{c}_2, \cdots, \widetilde{c}_K\}$. Then the accuracy is defined as

$$\text{Accuracy}(C, \widetilde{C}) = \frac{1}{M} \sum_k \max_j \left|\widetilde{c}_k \cap c_j\right|, \tag{1}$$

where $\text{Accuracy}(C, \widetilde{C})$ is more concerned with the accuracy within each cluster. This is because for DNA storage, if the sequences within a cluster belong to different original sequences, it will seriously affect the global comparison and subsequent data processing. In addition, we will discuss other metrics such as *Coverage* and *Redundancy Rate* to measure the quality of clustering.

We will next prove that the computational complexity of Clover is linear. This property is based on the characteristics of the tree structure and Clover's data processing flow.

Theorem 1.7. (*Computational Complexity*) The computational complexity of the Clover algorithm is $O(M)$. If the global matching function is enabled, the theoretical number of global matches is $(M - N)$, and cannot exceed M.

Proof We first prove that the computational complexity is $O(M)$ for M sequences to be classified. As can be seen from Algorithm 2 that Clover only needs to loop M times to cluster them. Although node drift sometimes increases the number of

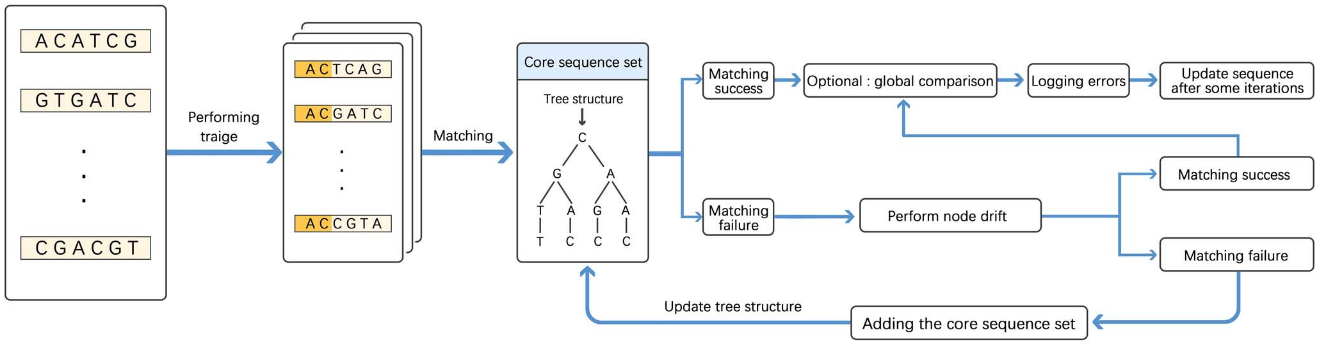

Figure 4. The overall flow chart of the Clover algorithm with a multi-process and global comparison function, where the number of processes is 16 and the dataset will be triaged according to the first 2 bases.

retrievals, the number of node drifts that can be generated per sequence is limited and the upper limit of node drift is not affected by the increase of M. Therefore, it is clear that our computational complexity is $O(M)$.

Due to the nature of our algorithm, any sequence that enters Clover for clustering will give a result of either matching one of the core sequences or being added to the core set as a new core sequence. We assume that a total of c clusters are generated after clustering, i.e. there are c sequences in the core sequence set. As sequences are added to the core sequence set no global comparison is required. Therefore, there are C sequences that will not be compared globally. The remaining $M - C$ entries will then either be discarded (e.g. due to broken chains resulting in too short a sequence length) or clustered into clusters. If the sequences are discarded then no global comparison will be performed. If they are clustered into clusters they will only be compared with the core sequences, and thus only once. The size of the total number of comparisons A satisfies $A \leqslant M - C < M$.

Theoretically, if N clusters are successfully recovered and all sequences are accurately identified, then $C = N$. The number of comparisons is $A = M - C = M - N$. ■

Not only does Clover have linear computational complexity, but its space complexity is also only related to the number of original clusters and not to the total number of sequences. We next give the space complexity of Clover, whose proof relies on an extreme case discussion for the number of tree nodes, since it is the nodes of the tree that mainly occupy memory in Clover.

Theorem 1.8. (*Space Complexity*) The theoretical maximum memory footprint of this algorithm is $O(4^t)$. However, if the maximum occupancy is reached, the algorithm fails at this point. In fact, the maximum memory usage of this algorithm is $O(N)$.

Proof We first show that the limiting case has a space complexity of $O(4^t)$. Since the tree has at most four branches, the number of nodes in the tree structure at depth t is at most four, so the space complexity is $O(4^t)$. However, due to the nature of the Clover algorithm, when the tree structure is filled, the algorithm will lose its clustering effect, so the actual space complexity will be much lower than $O(4^t)$. ■

Before proving the actual computational complexity, we first prove a lemma.

Lemma 1.9. For a tree structure of depth t, if N sequences are filled in, the number of nodes Q in the tree is

$$\frac{4^{\lceil \log_4 N \rceil + 1} - 1}{3} + (t - \lceil \log_4 N \rceil - 1) + N - 4^{\lceil \log_4 N \rceil} \leq Q$$
$$\leq \frac{4^{\lceil \log_4 N \rceil + 1} - 1}{3} + (t - \lceil \log_4 N \rceil) N - 4^{\lceil \log_4 N \rceil} \qquad (2)$$

Proof Due to the characteristics of the tree structure it is clear that the earlier any two tree branches are offset the more nodes they occupy, while the later the branches occur the fewer nodes they occupy. Figure 5 shows schematic representations of the theoretical maximum and minimum occupancy of the n-sequence, respectively.

We first consider the case of the maximum number of nodes. In that case, we require the tree to quadruple fork from the vertex up to the x-th level satisfying: $4^{x-1} < N \leqslant 4^x$. Thus, x can be expressed as $x = \lceil \log_4 N \rceil$, where $\lceil\ \rceil$ indicates an upper rounding. The number of nodes in the first x layers is

$$\frac{4^{\lceil \log_4 N \rceil + 1} - 1}{3} - \left(4^{\lceil \log_4 N \rceil} - N\right). \qquad (3)$$

Since there will be no overlapping nodes in each sequence in the latter $t - x$ layers, the number of nodes in the latter $t - x$ layers is $(t - \lceil \log_4 N \rceil - 1) N$.

Considering the practical case that the number of joinable entries in the tree is much greater than n at this point, $t-x$ must be greater than zero. Thus, the maximum number of nodes produced by the n-th sequence is

$$\frac{4^{\lceil \log_4 N \rceil + 1} - 1}{3} + (t - \lceil \log_4 N \rceil) N - 4^{\lceil \log_4 N \rceil}. \qquad (4)$$

Next, we consider the minimum number of nodes. Similar to the maximum number of nodes, we need to concentrate the forks of the tree at a certain interval level. But in contrast to the maximum number of nodes construction, we try to fork at the end of the tree to reduce the number of nodes generated. So for the value of x found in the maximum number of nodes, we will have only one node at each of the first $t - x$ layers of the tree, and fork from the $t - x + 1$ layer. Thus, the number of nodes in the first $t-x$ layers is $(t - \lceil \log_4 N \rceil - 1)$. The number of nodes in the second x layers is

$$\frac{4^{\lceil \log_4 N \rceil + 1} - 1}{3} - \left(4^{\lceil \log_4 N \rceil} - N\right). \qquad (5)$$

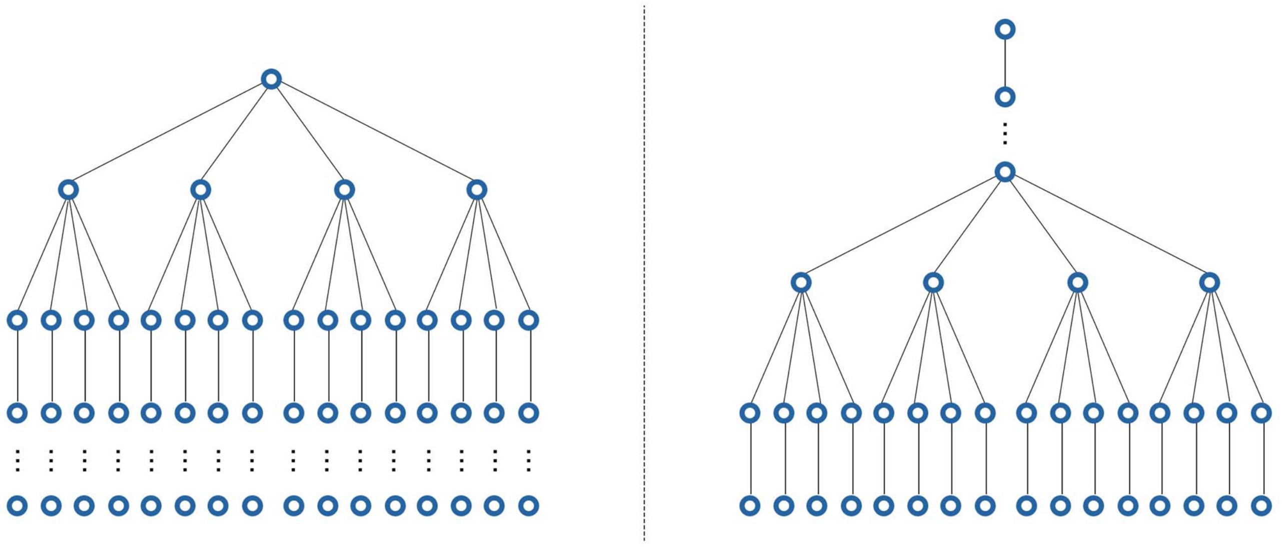

Figure 5. Extreme case node situation.

The minimum number of nodes is calculated by

$$\frac{4^{\lceil \log_4 N\rceil+1}-1}{3}+\left(t-\lceil \log_4 N\rceil-1\right)+N-4^{\lceil \log_4 N\rceil}. \quad (6)$$

The memory footprint of Clover is mainly divided into the core sequence set and the tree nodes, where the core sequence set is shown as a linear complexity of about N. Based on the above citation, we can learn that when the parameter t is fixed, the maximum number of nodes is influenced by N and shows linear complexity. Therefore, the space complexity of Clover is $O(N)$. ■

Based on the previous definition of accuracy, we give the theoretical error rate of Clover.

Theorem 1.10. (*Error Rate*) We assume that the effect of pre-sequence base errors on post-sequence bases is not considered (node drift can resist the effect caused by pre-sequence base errors). The probability that the sequence cannot be successfully matched to the original sequence is

$$P \le \left[1-\left(1-e^{-\frac{(x-\varepsilon)^2}{2\varepsilon}}\right)(1-P_d-P_i)^t\right]^N. \quad (7)$$

Proof We first consider the impact of matching due to substitution errors. Since when there are more than x errors, the node drift will not be corrected back to the original sequence at this point, and the match is incorrect at this point. Since the probability of an error at each position is p_s, the expectation of an error for an interval of length t is $tp_s = \varepsilon$.

Because of the characteristics of node drift as well as substitution errors, an error in one base will not affect subsequent base comparisons. Therefore, by Chernoff Bound, it is known that

$$P_\delta = P_r[E \geqslant (1+\delta)\varepsilon] < e^{-\varepsilon\frac{\delta^2}{2}}, \quad \forall\delta > 0. \quad (8)$$

Specifying that the comparison fails when $E \geqslant x$, we have

$$(1+\delta)\varepsilon = x \Rightarrow \delta = \frac{x}{\varepsilon}-1. \quad (9)$$

Thus, the probability of match failure due to base substitution is $P_A \leqslant e^{-\varepsilon\frac{(x-\varepsilon)^2}{2\varepsilon^2}} = e^{-\frac{(x-\varepsilon)^2}{2\varepsilon}}$.

Next, we discuss the effect of base deletions and base additions on sequence alignment. Since both deletions and additions produce substantial changes in the position of clips in the sequence, the premise that an interval can be successfully matched must be that no base additions and deletions occur, and therefore must satisfy: $P_N = (1-P_d-P_i)^t$.

In summary, the probability that an interval will be successfully matched to the original interval is

$$P \ge \left(1-e^{-\frac{(x-\varepsilon)^2}{2\varepsilon}}\right)(1-P_d-P_i)^t. \quad (10)$$

In addition, for n intervals of a sequence, only one interval needs to be matched to guarantee a successful sequence match, so the probability that the sequence cannot be successfully matched to the original sequence is

$$P \le \left[1-\left(1-e^{-\frac{(x-\varepsilon)^2}{2\varepsilon}}\right)(1-P_d-P_i)^t\right]^N. \quad (11)$$

■

Based on the above theorems, we prove that the Clover algorithm has theoretically linear computational complexity, its memory consumption is only related to the original data and the size of the tree and it has a very low theoretical error rate. Finally, we conduct experiments with real-world data to verify the above statement.

Experimental design

Dataset information

A total of four real-world datasets as well as one simulated dataset were used for experiments and their details are as follows.

- **PE-YAB** [4]: Goldman *et al.* [4] proposed a practical use of DNA storage in their pioneering work. They synthesized 153 335 DNA molecules, each of length 117, in which

Table 1. Statistical details of the datasets

Datasets	Read count	Original sequences count	Read length	Data sources
ERR1816980	14 654 644	72 000	152	[5]
ERR1817036	34 095 791	72 000	152	[5]
PE-YAB	73 688 127	153 334	117	[4]
P10-5-BDDP210000009	16 217 014	187 973	200	[29]
i18-s3-r1-001	15 169 628	16 383	60	[18]

the first and last two bases contain no information. The DNA was synthesized using Agilent's Oligo Library Synthesis (OLS) Sureprint technology and sequenced using Illumina HighSeq 2000. Here, we selected the sequencing file PE-YAB in the text as the experimental dataset.

- **ERR1816980** and **ERR1817036** [5]: Erlich and Zielinski [5] proposed a fountain code-based DNA coding technique that will allow the amount of information recoverable to be several orders of magnitude higher than before. They synthesized 152 long 72 000 DNA molecules. The DNA synthesis was performed using Twist Bioscience technology and the sequencing was performed using Illumina Miseq V4 technology. Here, we selected the sequencing files ERR1816980 and ERR1817036 as the experimental datasets.
- **P10-5-BDDP210000009** [29]: Song *et al.* [29] proposed a new approach for DNA storage by synthesizing 210 000 DNA sequences while storing a 6MB file. Here, we selected the sequencing file P10-5-BDDP210000009 as the experimental dataset.
- **I18-S3-R1-001** [18]: Antkowiak *et al.* [18] proposed a DNA storage system that relies on massively parallel light-directed synthesis. They used a light-directed maskless array technique to synthesize DNA to reduce costs, where 16 383 sequences at a length of 60 were synthesized. Experiments showed that the measured error probabilities were 2.6% for substitutions, 6.2% for deletions and 5.7% for insertions. This is currently the highest error rate dataset in the DNA storage field. Here, we selected the sequencing file I18-S3-R1-001 as the experimental dataset.
- **Simulated dataset**: To evaluate the clustering of Clover for a very large dataset, we constructed a simulated dataset containing 10 billion sequenced sequences. The simulated dataset has 10 million original sequences, which are evenly amplified to 10 billion sequences, and each base position has a 1 in 300 chance of having an addition, deletion or substitution error when amplified (all three errors occur with equal probability). The entire dataset was generated using the python language and the random module.

Table 1 shows the specific details of each dataset. The real-world data from four different sources contain the latest DNA storage synthesis data that can be found and the photosynthesis DNA data with a high error rate, which can improve the validity of the algorithm evaluation.

Producing high confidence labeled datasets

In order to be able to obtain labeled datasets to test the accuracy of the clustering, we compared the above datasets with its original reference set to produce a highly confident labeled dataset. Both bioinformatics software, pear and bowtie2, were used for the comparison process. The details of the comparison are as follows.

Paired-end aggregation

Paired-End reAd mergeR (PEAR) [30] is a fast, low memory and highly accurate read merge software for pairs of ends. For data generated from both ends, we first use PEAR to perform a double-ended merge.

Sequence matching

We then compare the merged sequences to the reference sequence set by using Bowtie2 [31], which is a fast and efficient sequence alignment tool. It is particularly good at matching shorter sequences to reference sequences and is therefore well suited to the matching of stored DNA sequences. We first build the index using the reference sequence set.

The pear-merged sequence set seq-assembled.fastq was later used for comparison. Since the real-world datasets PE-YAB, ERR1816980, ERR1817036 and P10-5-BDDP210000009 have a relatively low error rate, we use the default pattern of bowtie2 for clustering. Since I18-S3-R1-001 is photochemically synthesized and has a high error rate, bowtie2's default mode does not compare well, so we used local alignment mode and reduced the length of the seed substring to make bowtie more sensitive to alignment. It is important to note that, unlike other datasets where the majority of sequences can be matched successfully, the photosynthetic DNA sequences have a very high error rate, resulting in only about half of the sequences being matched even when the bowtie2 parameters are adjusted to be extremely sensitive (we only consider sequences around 60bp in length).

In order to avoid reducing the clustering difficulty by filtering part of the data, for this dataset, we will input all sequences around 60bp into the Clover algorithm for clustering, regardless of whether these sequences have a label or not, thus enhancing the validity of our experiment. However, for unlabeled sequences, we will ignore the effect on accuracy when performing post-clustering statistics (this is unavoidable, as it is not easy to determine the label of a sequence if the original sequence cannot be identified by comparison with the original set of sequences. This is not a flaw in our clustering algorithm). In addition, we also used bowtie's default comparison parameters to generate a smaller dataset (about 420 000 sequences) for comparison of the accuracy of the different algorithms.

Data processing

We use the SAM file output in bowtie2 for further data processing. We use python to process the data, where the labels and sequences are extracted to construct a dataset with labels in txt format, where each row is Label Sequence. In addition, we also perform the error rate statistics in Appendix using the CIGAR values from the SAM file as a reference.

Table 2. Equipment information

Type	Name	CPU	Memory	Equipment Provider
Desktop computer	PC	AMD Ryzen 5 3600 (6 processors)	64G	-
Cloud Server	VM-21E-0ZU41	Intel 6242R 3.1GHz (32 processors)	128G	Yovole Cloud
Supercomputer Server	$D2Part_{test}$	Intel 6258R 2.7GHz (112 processors)	3T	National Supercomputing Center

Table 3. Clover function

Type	Datasets	Input	Output	Parallel mode	Low memory mode	Global comparison mode
Virtual mode	Labeled datasets	SAM, TXT	Time, Accuracy, Coverage	✓	×	✓
Actual mode	Unlabeled datasets	FASTA, FASTQ, TXT	Time, Clusters, Core dataset	✓	✓	✓

Experimental setup

To simulate different experimental environments, three devices are used in the experiments: a normal desktop computer, a cloud server and a supercomputer, the detailed parameters of which are shown in Table 2. In the experiments, all real-world datasets are run on the cloud server, which is also the most common experimental environment. In addition, the simulated datasets are executed in single-threaded mode on a regular desktop computer to demonstrate the applicability of Clover in extreme environments, and in parallel mode on a supercomputer to demonstrate the extreme performance of Clover.

Clover is a practical and efficient DNA sequence clustering algorithm that allows the user to customize multiple parameters to meet different needs. Clover has two modes, one is for inputting unlabeled sequences, in which Clover allows the output of label classes for each sequence as well as the core set of sequences. The other mode is for input sequences with labels, where Clover allows the output of the labels, core set and accuracy of the sequences to evaluate Clover's clustering effectiveness. Clover also allows the use of multiple processes to improve the clustering speed, but the number of processes must be an exponent of 4. We do not recommend using too many processes as this will increase the number of clusters after clustering. Clover is written in Python to allow users to embed custom global matching algorithms; experiments will demonstrate that it still has a very fast clustering speed. Clover also has a low-memory mode, in which Clover will output clustering results directly after each input sequence has been clustered, thus reducing the memory footprint, but this mode does not support outputting accuracy information, although this is not affected in practice. Moreover, Clover supports other usage scenarios, including the identification and compatibility with other types of bases in the sequence, which are documented in detail in the code documentation. The specific features of Clover are given in Table 3.

As the length and structure of the sequences in different real-world datasets vary, we use different Clover parameters to adjust accordingly. Overall, the depth of the tree will be around 15–20, as too shallow a depth will reduce accuracy, while too high a depth will increase memory significantly. In terms of the number of trees, we will have at least two trees at the front and back ends, and one or two trees in the middle of the sequence. Table 4 shows the detailed parameter settings for different datasets.

We choose CD-HIT [21], MeShClust [22], DNACLUST [32], Starcode [24], UCLUST [25] and DBSCAN as the benchmark algorithms, and they are the current sequence clustering algorithms with excellent performance. The identity score threshold of all algorithms was set to 0.9, where CD-HIT, DNACLUST, Starcode and MeShClust used 16 threads, DBSCAN used single threads and UCLUST used 10 threads due to memory constraints.

Scalability analysis

We extract different numbers of datasets for the ERR dataset as benchmark datasets to demonstrate the clustering effect more quantitatively.

Figure 6 shows a time-consuming comparison of different clustering algorithms. It can be seen that the clustering speed of Clover is faster than other benchmark algorithms. The multi-process mode of Clover can speed up clustering by a factor of about 10. Although all benchmark algorithms except DBSACAN use multi-process mode, their clustering speed is still lower than that of single-threaded Clover.

Figure 7 shows the memory usage of the algorithm for the ERR1817036 dataset. It can be seen that the memory grows rapidly at the beginning, and then grows steadily. This is mainly due to the need to input a large number of sequences into the core sequence set to build a tree structure in the early stage, while the core sequence set basically stabilizes at the later stage, and the addition of tree structure becomes less. It can be observed from this figure that more memory is occupied after using multiple processes. This is mainly due to the overlapping of tree structure branches in different processes when the data are shunted. Overall, for the clustering of 70 million DNA sequences, Clover can guarantee an accuracy rate of over 97% with minimum memory consumption of 0.6G, which proves that the memory consumption of Clover is very low.

Accuracy analysis

Table 5 shows the accuracy of Clover with different datasets. It can be seen that Clover has very high accuracy on all four real-world datasets, especially on the ERR1816980, ERR1817036 and P10-5-BDDP210000009. PE-YAB does not achieve very high accuracy, mainly because its encoding method makes the original sequences closer to each other, which increases the possibility of errors. Note that the i18-s3-r1-001 has a 10 times higher error rate than the others, which is the highest error rate we could find for the real-world dataset. Compared with turning on the global matching mode, the performance of Clover is not significantly degraded, so the clustering accuracy of Clover does not depend on the global comparison between sequences. In this case, we can still maintain a high accuracy of over 97%, proving that Clover still has a strong clustering ability for datasets with high error rates.

Table 4. Clover parameter setting

Datasets	Number of trees	Depth of tree	Horizontal drift values	Vertical drift values
ERR1816980	4	15	3	3
ERR1817036	4	15	3	3
PE-YAB	3	20	1	3
P10-5-BDDP210000009	4	20	3	3
i18-s3-r1-001	4	20	1	3
Simulated dataset (low memory mode)	3	15	3	3
Simulation of dataset (parallel mode)	4	20	3	3

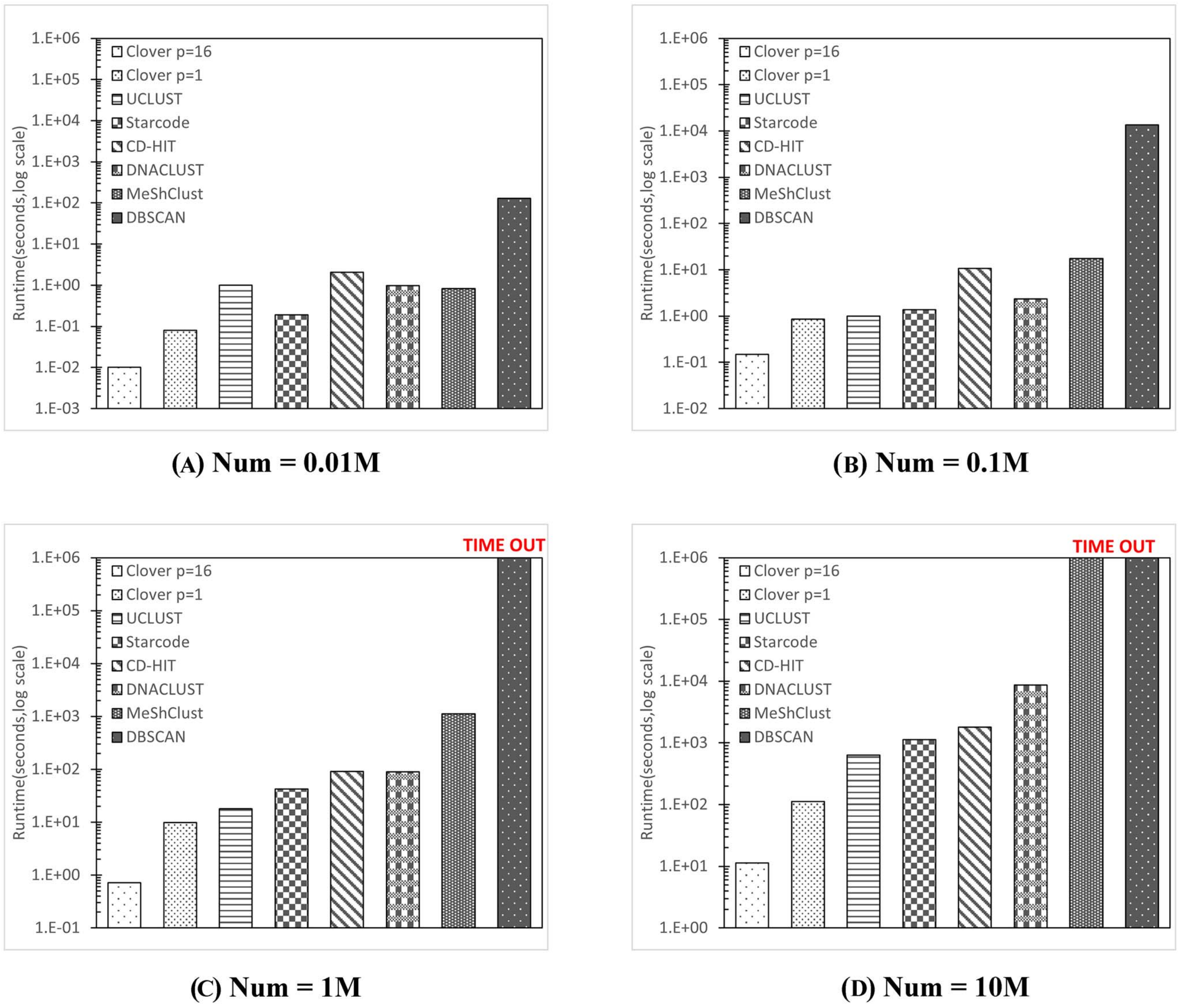

Figure 6. A time-consuming comparison of different clustering algorithms on datasets of different orders of magnitude. Different parameter settings of Clover will have a certain impact on the runtime, but they are all lower than the time-consuming of the benchmark algorithms. The results that take more than 10 h are marked as 'time-out' in red.

Table 6 shows the comparison of the results of different clustering methods under the i18-s3-r1-001 small dataset. It can be seen that Clover has lower time consumption than other benchmark methods. Although Starcode is close to Clover in terms of time consumption, its accuracy is only 23.17%, and it does not complete the clustering very well. In terms of accuracy, despite the high base error rate in the dataset, the accuracy of both Clover and CD-HIT can exceed 99.9%. MeShClust has no accuracy because it cannot be successfully clustered. Furthermore, we also compare the number of clusters after clustering, and since the original clusters of the dataset are about 10 000, we want the number of clusters after clustering to be as close as possible to the original number of clusters. The table shows that the number of clusters of Clover is much smaller than the other benchmark methods

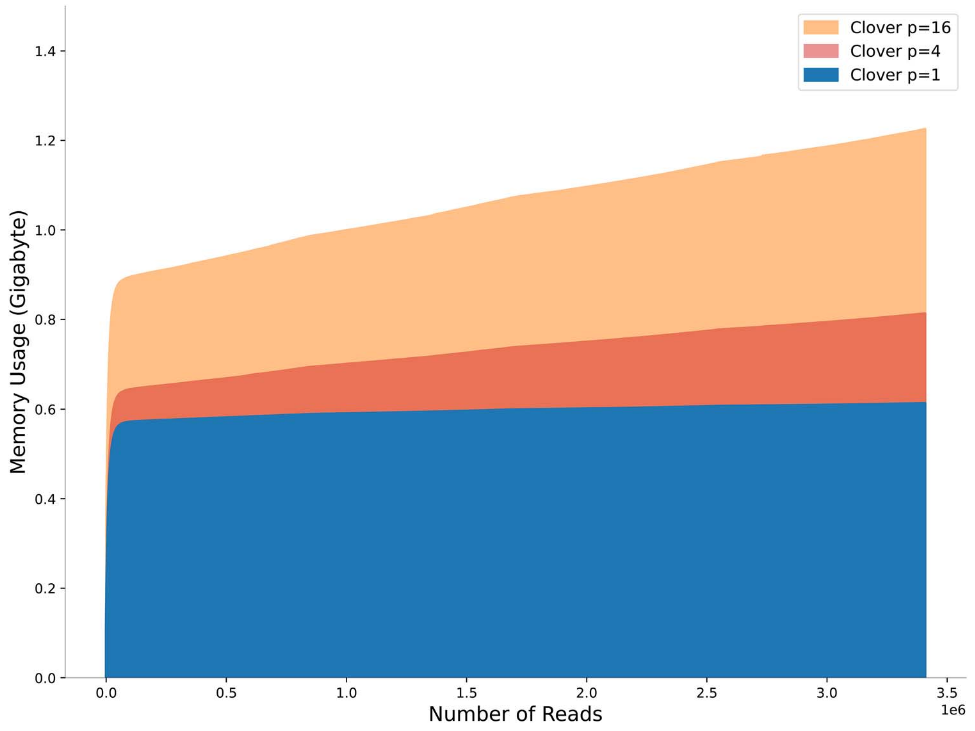

Figure 7. Memory usage graph. The abscissa is the number of sequences read and the ordinate is the memory usage. It can be seen that the memory usage increases sharply at first and then stabilizes, and also increases as the number of processes increases.

Table 5. Accuracy comparison under various datasets

Dataset	Clover $p = 1$	Clover $p = 1$ GM	Clover $p = 4$	Clover $p = 4$ GM	Clover $p = 16$	Clover $p = 16$ GM
0.01M	100%	100%	100%	100%	100%	100%
0.1M	99.99%	99.93%	99.99%	99.99%	99.99%	99.99%
1M	99.93%	99.93%	99.99%	99.99%	99.99%	99.99%
10M	99.52%	99.5%	99.93%	99.93%	99.94%	99.94%
ERR1816980	99.35%	99.33%	99.9%	99.9%	99.92%	99.92%
ERR1817036	99.41%	99.41%	99.89%	99.91%	99.93%	99.95%
PE-YAB	97.64%	97.33%	97.79%	97.8%	97.74%	97.74%
P10-5-BDDP210000009	99.87%	99.87%	99.87%	99.87%	99.87%	99.87%
i18-s3-r1-001	97.86%	97.86%	97.25%	97.25%	97.49%	97.49%

and closer to the original number of clusters, because Clover will discard garbage sequences and generate clusters in a strict way.

Stability analysis

This subsection will give the stability performance of Clover under different parameters. We will give the differences in time consumption, accuracy and memory usage of Clover for different tree depths in real datasets. In addition, we will test the computational complexity of Clover and its effectiveness in the limit state by clustering a large-scale simulated dataset.

Figure 8 shows the effect of different tree depths on the clustering effect. Figure 8A shows that the clustering time increases as the tree gets deeper, but if the tree depth is too shallow, it will increase the number of drifts generated during clustering, which in turn will increase the time consumption. The memory consumption is shown in Figure 8B. Due to the characteristics of the tree structure, the memory consumption increases more than linearly with the depth of the tree, but generally we do not set the depth of the tree to be particularly large. This is because as shown in Figure 8C, a depth of 15 can achieve very high accuracy in terms of accuracy, especially for large amounts of data.

To demonstrate the advantage of Clover's clustering model for extremely large-scale datasets, we simulate 10 billion DNA sequences, the largest known dataset in the DNA storage domain. To verify the ease of use of Clover, we conduct experiments on a large-scale simulated dataset on a home computer.

Figure 9 shows the correlation between dataset size and elapsed time, and it can be seen that our algorithm retains almost linear computational complexity under extremely large-scale datasets. We then conduct multi-threaded experiments on a supercomputing server, and the experimental results show that Clover can cluster 10 billion DNA sequences in about 4 h with a clustering accuracy of 99.99% under multiple processes. This demonstrates that Clover can still cluster large-scale datasets in a short time with high accuracy and excellent hardware friendliness.

Table 6. Comparison results of different clustering algorithms

Metrics algorithms	Clover	UCLUST	CD-HIT	DNACLUST	Starcode	MeShClust
Runtime	**1.3s**	25s	61.21s	237.1s	4.2s	2737.3s
Accuracy	99.94%	98.53%	**99.96%**	85.99%	23.17%	-
Number of cluster	**49 617**	195 035	264 212	143 846	357 127	425 711

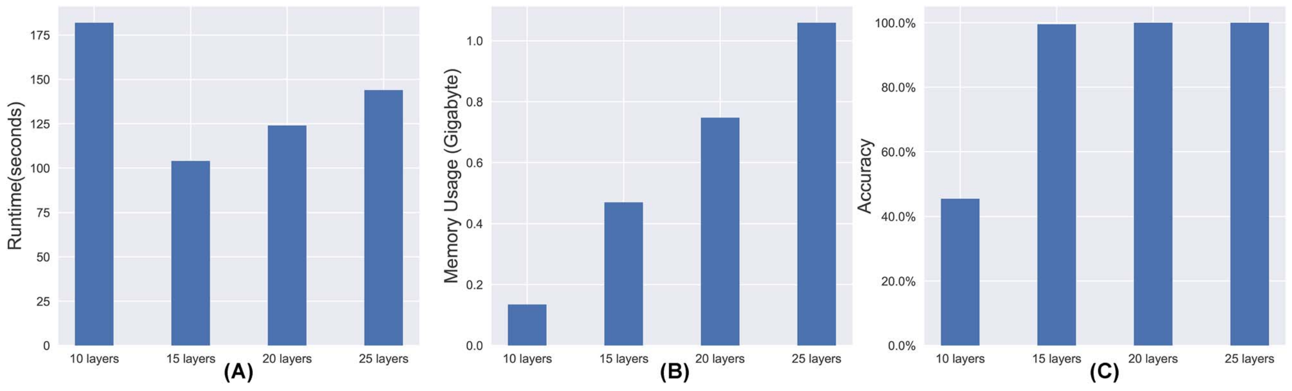

Figure 8. The effect of different depths. **(A)** shows the effect of different depths of the tree on the time consumption. **(B)** shows the effect of different depths of the tree on memory consumption. **(C)** shows the effect of different depths of the tree on the accuracy rate.

Discussion on coverage and redundancy

With our definition of accuracy, there is a theoretical extreme case where we add every sequence as a class to the core set, then by our definition of accuracy, it is still 100%, as there are no clusters with different labels within each cluster. To avoid this, Clover has a built-in parameter R. If the final output cluster has less than R sequences in it, the cluster is simply discarded. It is known that reducing logical redundancy can lead to a high probability of decoding failure [33, 34]. However, if R is too large, many valid clusters will be lost, so we will focus on the effect of different values of R on the clustering of Clover. We first give the definition of Coverage and Redundancy, where Coverage reflects the inclusion of clustering results for the original clusters and Redundancy Rate reflects the number of noisy clusters in the clustered clusters.

Definition 1.11. *(Coverage)*: If there are N original sequences and the final output core set of sequences belongs to $\bar{N}$ different original sequences, the coverage is $\frac{\bar{N}}{N}$.

Definition 1.12. *(Redundancy Rate)*: If there are N original sequences and the number of sequences in the final output core sequence set is $\bar{M}$, then the redundancy rate is $\frac{\bar{M}-N}{N}$.

From the definition of coverage, we can see that a reduction in coverage will leave some of the original sequences missing. Although most current DNA storage coding algorithms overcome the problem of partial DNA sequences by using multiple layers of coding, we still want to ensure as high a coverage as possible. However, we do not want to have a very large redundancy rate, as this would increase the time for subsequent DNA decoding. We will then begin with a theoretical discussion of coverage, followed by an experimental evaluation on coverage and redundancy.

Theoretical analysis of coverage

We first give some assumptions. After the clustering is completed, we will discard the classes that contain less than d sequences, we assume that the clustering is finished with the original $\bar{M}$ sequences and after filtering is M sequences. For any original sequence, DNA synthesis and sequencing satisfy independent identical distribution and the distribution satisfies the Bernoulli distribution. That is, assume that the probability of being sampled is q. Assume that each sequence is sampled n times, and each success probability is q. And the different sequences are independent of each other. Let C_i denote the number of times x_i is sampled, we can find $C_i \sim B(n, q)$. Since we proved in the theorem that the Clover error rate is extremely low, we do not consider the effect of clustering errors.

It is known that all C_i are independently and identically distributed and satisfy $C \sim B(n, q)$, $\mathbb{E}(C) = nq$, $\mathbb{D}(C) = nq(1-q)$. Let D_d denote the number of sequences that are sampled d times. Thus, it is obtained that

$$\bar{M} - \sum_{d=0}^{K} D_d = M. \tag{12}$$

After rearranging, we can further obtain

$$\frac{M}{\bar{M}} = 1 - \frac{1}{\bar{M}}\sum_{d=0}^{K} D_d = 1 - \frac{1}{\bar{M}}\sum_{d=0}^{K}\sum_{i=1}^{\bar{M}} \mathbf{Pr}\left[C_i = d\right]$$
$$= 1 - \sum_{d=0}^{K} \mathbf{Pr}[C = d] = 1 - \mathbf{Pr}[C \leqslant k]. \tag{13}$$

Using the central limit theorem, it is obtained that

$$P_t[C = d] = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(d-\mu)^2}{2\sigma^2}}, \quad C \sim N(nq, nq(1-q)). \tag{14}$$

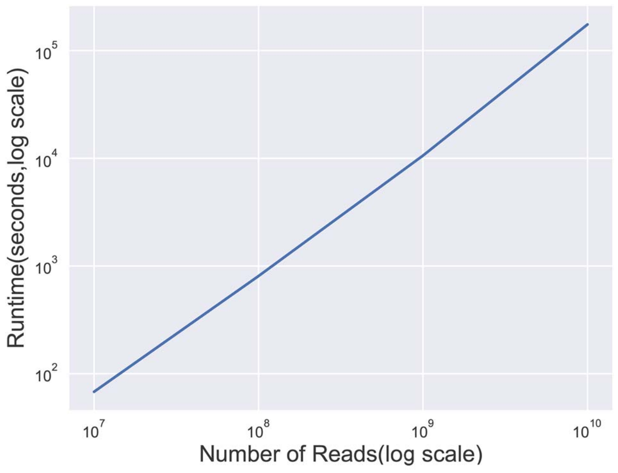

Figure 9. Time-consuming growth graph for different orders of magnitude of data.

The standardization leads to

$$\Pr[C \le k] = \Pr\left[c^* \leqslant \frac{k-\mu}{\sigma}\right],\ \mu = nq, \sigma = \sqrt{nq(1-q)}. \tag{15}$$

Therefore, we can query the standard normal distribution table to obtain the theoretical coverage. The above theoretical analysis can show that Clover has high coverage in theory, as long as the parameter R is not set too high.

Experimental analysis of coverage and redundancy

We selected a real-world dataset of 10 million entries as the experimental dataset, which is derived from ERR1816980 and has 49 119 clusters. We performed clustering using Clover with different R and the number of processes, respectively, and calculated the coverage and redundancy rates.

Figure 10 shows the effect of the choice of R, from which it can be seen that as the number of processes increases, the redundancy rate will also increase. In particular, at low R, due to the fact that some of the sequences with errors in the front end will be incorrectly shunted, which in turn leads to an increase in noisy clusters. However, the increase in the number of processes can weakly increase the coverage rate, which is due to the decrease in the number of original clusters in each process, reducing the probability of a cluster being misclassified. In addition, this figure shows that the coverage and redundancy decrease as R increases, which is obvious because as the range of discarding becomes larger, more real clusters are discarded with noisy clusters, which in turn reduces the coverage and redundancy.

In summary, although the clustering effect of Clover is affected by the coverage and redundancy, by choosing a suitable R, we can obtain a reasonable coverage and redundancy, and the effect caused by discarding some clusters will hardly affect the decoding work after clustering.

Conclusion

We present Clover, a novel DNA clustering method based on a tree structure, which avoids a large amount of time spent on computing the Levenshtein distance by retrieving the tree. We theoretically prove that Clover has linear computational complexity and low space complexity. Furthermore, we experimentally verify its efficient clustering speed and small memory footprint since it clusters an unprecedentedly large dataset on a home computer. For future work, we will further improve the algorithm's ability to identify redundant clusters for solving the sequence reconstruction problem in DNA data storage [35, 36], and extend its application scenarios beyond the DNA storage domain.

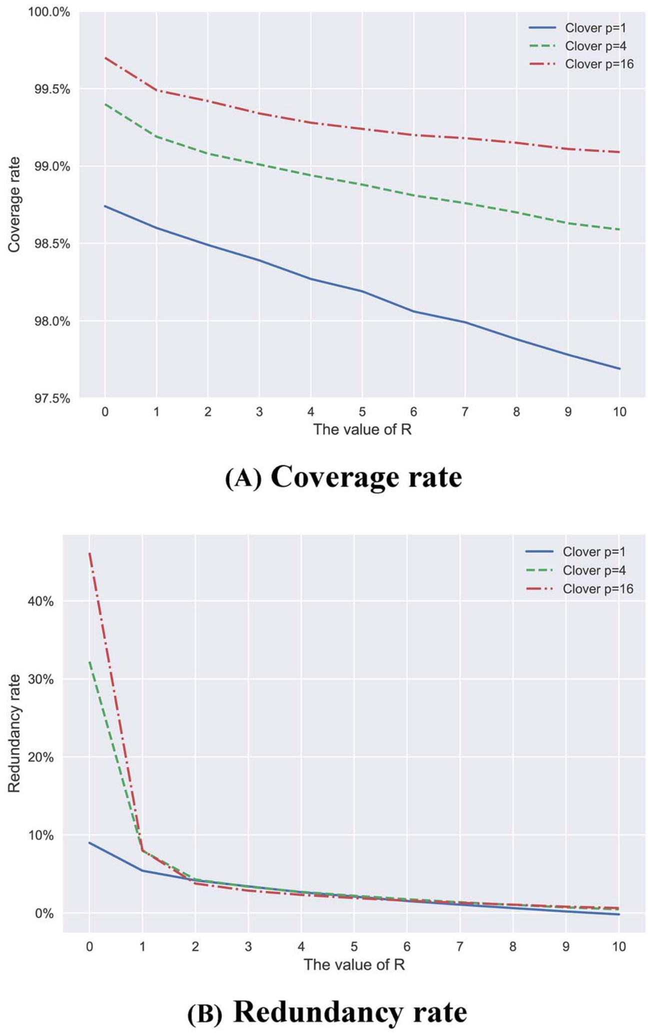

Figure 10. The effect of different processes and R on coverage and redundancy.

Key Points

- We design Clover, a DNA sequence clustering method dedicated to DNA data storage with linear complexity, low memory consumption and parallelism.
- We demonstrate the linear complexity and high accuracy of Clover. We also experimentally verified that it is more than 10 times faster than existing DNA clustering methods.
- Clover can be applied to cluster 10 billion sequences with an accuracy of 99.99%, an order of magnitude that has not been studied before. Moreover, we successfully clustered 10 billion sequences using a single home computer, demonstrating the applicability of Clover.

Supplementary data

Supplementary data are available online at https://github.com/Guanjinqu/Clover.

Author contributions statement

G. Q. and H. W. conceived and designed the study. G. Q. conducted the experiments and wrote the manuscript. G. Q. and Z. Y. analyzed the theoretical results. H. W. reviewed the manuscript.

Funding

This work is supported by the National Key Research and Development Program of China (# 2020YFA0712102) and the National Natural Science Foundation of China (# 62071327).

References

1. Tavella F, Giaretta A, Dooley-Cullinane TM, *et al.* Dna molecular storage system: Transferring digitally encoded information through bacterial nanonetworks. *IEEE Trans Emerg Top Comput* 2019;**9**(3):1566–80.
2. Ebrahimi S, Salkhordeh R, Osia SA, *et al.* Rc-rnn: Reconfigurable cache architecture for storage systems using recurrent neural networks. *IEEE Trans Emerg Top Comput* 2021;1–1.
3. Church GM, Gao Y, Kosuri S. Next-generation digital information storage in dna. *Science* 2012;**337**(6102):1628–8.
4. Goldman N, Bertone P, Chen S, *et al.* Towards practical, high-capacity, low-maintenance information storage in synthesized dna. *Nature* 2013;**494**(7435):77–80.
5. Erlich Y, Zielinski D. Dna fountain enables a robust and efficient storage architecture. *Science* 2017;**355**(6328):950–4.
6. Dong Y, Sun F, Ping Z, *et al.* Dna storage: research landscape and future prospects. *Natl Sci Rev* 2020;**7**(6):1092–107.
7. Lee O, Ang SD, Chen Y-J, *et al.* Random access in large-scale dna data storage. *Nat Biotechnol* 2018;**36**(3):242–8.
8. Jialu H, Zhong Y, Shang X. A versatile and scalable single-cell data integration algorithm based on domain-adversarial and variational approximation. *Brief Bioinform* 2022;**23**(1).
9. Cevallos Y, Nakano T, Tello-Oquendo L, *et al.* A brief review on dna storage, compression, and digitalization. *Nano Communication Networks* 2022;**31**:100391.
10. Grass RN, Heckel R, Puddu M, *et al.* Robust chemical preservation of digital information on dna in silica with error-correcting codes. *Angew Chem Int Ed* 2015;**54**(8):2552–5.
11. Hossein Tabatabaei Yazdi SM, Kiah HM, Garcia-Ruiz E, *et al.* Dna-based storage: Trends and methods. *IEEE Transactions on Molecular, Biological and Multi-Scale Communications* 2015;**1**(3): 230–48.
12. Rasool A, Qiang Q, Wang Y, *et al.* Bio-constrained codes with neural network for density-based DNA data storage. *Mathematics* 2022;**10**(5):845.
13. Smht Yazdi HM, Kiah E, Garcia-Ruiz J, *et al.* Dna-based storage: Trends and methods. *IEEE Transactions on Molecular, Biological and Multi-Scale Communications* 2015;**1**(3):230–48.
14. Alsaffar MM, Hasan M, McStay GP, *et al.* Digital dna lifecycle security and privacy: an overview. *Brief Bioinform* 2022;**23**(2):bbab607.
15. Heckel R, Mikutis G, Grass RN. A characterization of the dna data storage channel. *Sci Rep* 2019;**9**(1):1–12.
16. Zhang H, Lan Z, Zhang W, *et al.* Spider-web enables stable, repairable, and encryptible algorithms under arbitrary local biochemical constraints in dna-based storage. *arXiv preprint arXiv:220402855* 2022.
17. Jeong J, Park S-J, Kim J-W, *et al.* Cooperative sequence clustering and decoding for dna storage system with fountain codes. *Bioinformatics* 2021;**37**(19):3136–43.
18. Antkowiak PL, Lietard J, Darestani MZ, *et al.* Low cost dna data storage using photolithographic synthesis and advanced information reconstruction and error correction. *Nat Commun* 2020;**11**(1):1–10.
19. Hartigan JA, Wong MA. Algorithm as 136: A k-means clustering algorithm. *J R Stat Soc Ser C Appl Stat* 1979;**28**(1):100–8.
20. Ester M, Kriegel H-P, Sander J, *et al.* Density-based spatial clustering of applications with noise. *In Int Conf Knowledge Discovery and Data Mining* 1996;**240**:6.
21. Limin F, Niu B, Zhu Z, *et al.* Cd-hit: accelerated for clustering the next-generation sequencing data. *Bioinformatics* 2012;**28**(23):3150–2.
22. James BT, Luczak BB, Girgis HZ. MeShClust: an intelligent tool for clustering DNA sequences. *Nucleic Acids Res* 2018;**46**(14): e83–3.
23. Bao E, Jiang T, Kaloshian I, *et al.* Seed: efficient clustering of next-generation sequences. *Bioinformatics* 2011;**27**(18):2502–9.
24. Zorita E, Cusco P, Filion GJ. Starcode: sequence clustering based on all-pairs search. *Bioinformatics* 2015;**31**(12):1913–9.
25. Edgar RC. Search and clustering orders of magnitude faster than BLAST. *Bioinformatics* 2010;**26**(19):2460–1.
26. Edgar RC. Local homology recognition and distance measures in linear time using compressed amino acid alphabets. *Nucleic Acids Res* 2004;**32**(1):380–5.
27. Rashtchian C, Makarychev K, Rácz MZ, *et al.* Clustering billions of reads for dna data storage. In: Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, Roman Garnett (eds) *NIPS*. Red Hook, NY, USA: Curran Associates Inc., Vol. **2017**, 2017, 3360–71.
28. Jialu H, Chen M, Zhou X. Effective and scalable single-cell data alignment with non-linear canonical correlation analysis. *Nucleic Acids Res* 2022;**50**(4):e21–1.
29. Song L, Geng F, Gong Z, *et al.* Robust data storage in dna by de bruijn graph-based decoding. *bioRxiv*, pages 2020–12, 2021.
30. Zhang J, Kobert K, Flouri T, *et al.* Pear: a fast and accurate illumina paired-end read merger. *Bioinformatics* 2014;**30**(5):614–20.
31. Langmead B, Salzberg SL. Fast gapped-read alignment with bowtie 2. *Nat Methods* 2012;**9**(4):357–9.
32. Ghodsi M, Liu B, Pop M. Dnaclust: accurate and efficient clustering of phylogenetic marker genes. *BMC bioinformatics* 2011;**12**(1):1–11.
33. Heckel R, Shomorony I, Ramchandran K, *et al.* Fundamental limits of dna storage systems. In: *2017 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2017, 3130–4.
34. Ping Z, Chen S, Zhou G, *et al.* Towards practical and robust DNA-based data archiving using the yin–yang codec system. *Nat Comput Sci* 2022;**2**(4):234–42.
35. Srinivasavaradhan SR, Gopi S, Pfister HD, *et al.* Trellis bma: Coded trace reconstruction on ids channels for dna storage. In: *2021 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2021, 2453–8.
36. Sini MA, Yaakobi E. Reconstruction of sequences in dna storage. In: *2019 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2019, 290–4.

Appendix: Error distribution of DNA data

DNA is a long molecule consisting of four nucleotides (adenine, cytosine, guanine and thymine). Throughout the DNA storage process, the biological properties of nucleotides and the immaturity of current technology lead to a variety of errors that can occur during the synthesis of DNA, PCR amplification, DNA sequencing and other steps, including base addition, base deletion, base substitution and strand breakage. Therefore, studying the data characteristics and error distribution of DNA storage datasets will facilitate the improvement of algorithms involved in DNA storage, including coding and clustering algorithms. In the following text, we will present a detailed distribution of the different features of the DNA sequencing data using the real-world dataset P10-5-BDDP210000009 as a statistical dataset. P10-5-BDDP210000009 is the most recent DNA storage sequencing data available and the synthetic sequencing technology used is the mainstream technology, so it is a strong reference.

Sampling distribution after sequencing

The PCR process is stochastic and can be simplified by assuming that there are c homogeneous initial sequences in a sample pool and that each sequence in the pool is amplified with a certain probability p (becomes two) and not amplified with probability $1 - p$ (remains one) in each round of amplification. Furthermore, due to the specific nature of the sequencing process, it is not possible to sequence all the sequences in a sample to obtain its data, but often only a small number of sequences in a sequencing library, so the sequencing process is essentially a random sampling process. Thus, it is clear that there is a degree of randomness in both the amplification and sequencing processes, which will result in different numbers of sequences being sequenced from different original sequences in the final sequencing. The different numbers of sequenced raw sequences will affect the effect of DNA clustering and decoding, so it is important to assess the effect of randomness on the experimental results and to find a suitable mathematical algorithm.

Figure 11 shows the distribution of sequences in the 6MB dataset. We can see that the data exhibit an approximately normal distribution. Thus, we can use the normal distribution to simulate the sampling distribution of the original sequences after amplification and sequencing, which will have a strong reference for designing DNA storage algorithms.

Distribution of errors in the data

We first focus on sequence errors. We consider a sequenced sequence to be an error if it is not identical to its original sequence. Statistically, the erroneous sequences in P10-5-BDDP210000009 accounted for 14.67% of all sequences. Figure 12 shows the statistics of erroneous sequences for different original sequences. From the figure we can see that the distribution of the proportion of erroneous sequences for different raw sequences is an approximately normal distribution.

We also did an interesting statistic where we counted the proportion of error sequences in which similar errors occur. We defined that two error sequences are similar if they are identical. The total similarity error rate of the data was 34.93%, meaning that 34.93% of the erroneous sequences were similar sequences. We believe that similar error sequences are mainly caused by DNA synthesis or early PCR amplification. The study of similarity errors is useful to examine at which stage of DNA storage the errors

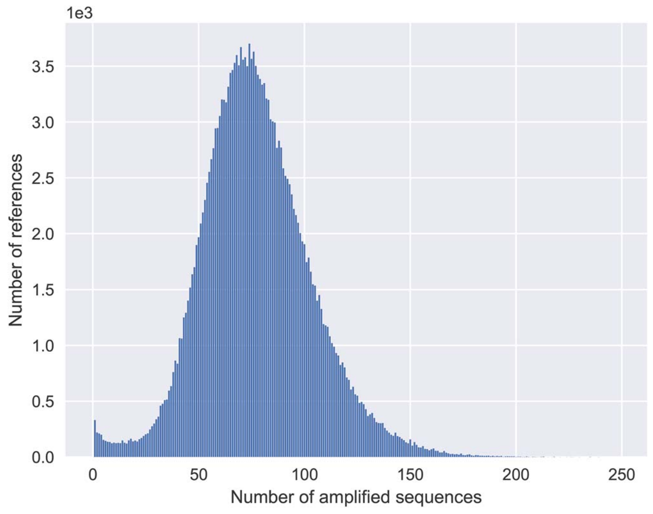

Figure 11. Reference amplification distribution. The horizontal coordinate indicates the number of sequences sequenced and the vertical coordinate indicates the number of sequences with that number of sequences.

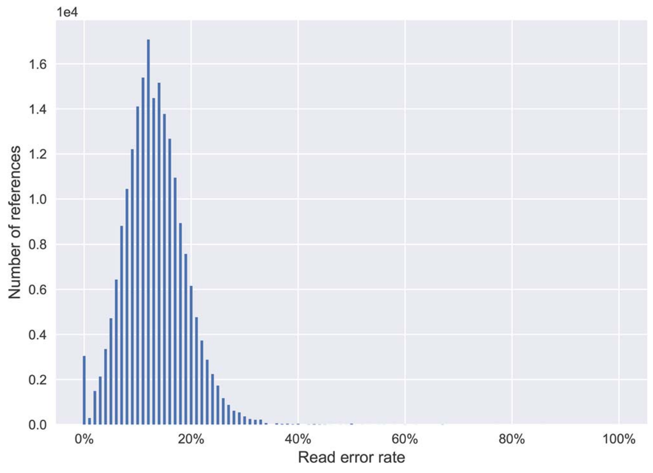

Figure 12. Error rate distribution. The horizontal coordinate indicates the percentage of erroneous sequences and the vertical coordinate indicates the number of raw sequences that have that percentage of erroneous cases.

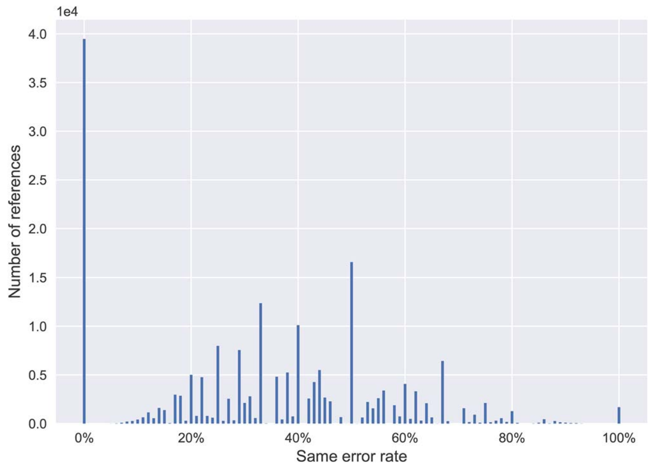

Figure 13. Similar error rate distribution. The horizontal coordinate is the similarity error rate and the vertical coordinate is the number of original sequences with that similarity error rate.

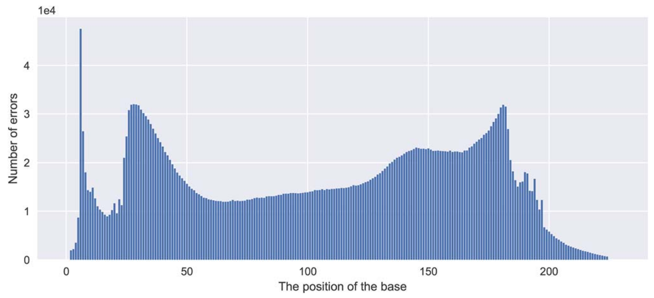

Figure 14. Base position error distribution. The horizontal coordinate is the position of the base and the vertical coordinate is the total number of base errors at that position.

are generated. Figure 13 shows the quantitative distribution of the similarity error rate. From the graph we can see that the distribution of the number of similarity error rates is less pronounced. This is mainly due to the fact that the similarity error rate is influenced by the total number of errors in the sequence, so for a raw sequence with fewer errors, the similarity error rate may be more extreme.

We next explore the base error errors at different positions. Figure 14 shows the number of base errors at different positions in the dataset. As can be seen from the graph, errors generally occur at the head and tail ends of the sequence, which can be well explained by the fact that the head and tail ends have a higher chance of error when performing synthesis and PCR amplification. In addition, the error rate increases nearer to the back end, which is also influenced by DNA synthesis. Note that our statistics are based on CIGAR values from bowtie comparisons and some

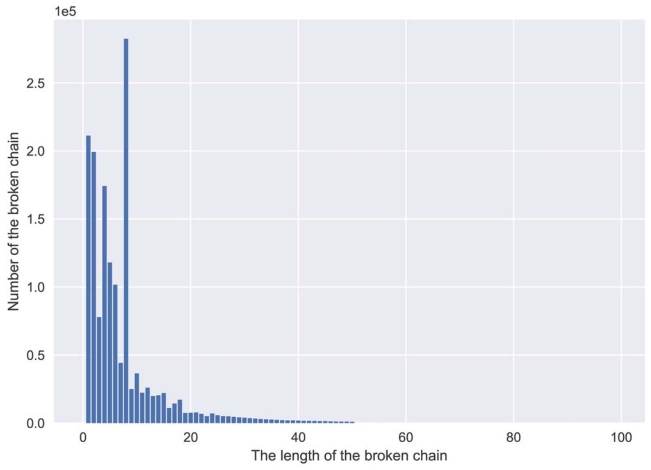

Figure 15. Broken chain length statistics. The horizontal coordinate is the number of bases mutilated by the broken strand and the vertical coordinate is the total number of broken strands of that length.

sequences with too many errors could not be compared and therefore could not be counted as errors.

We also explored the distribution of the number of broken chains. The broken chains are given in Figure 15. It can be seen that the number of broken strands becomes less as the strand is shortened. It can therefore be shown that the closer the bases are to the ends, the more likely they are to be lost. This provides an important reference for our subsequent DNA coding design (e.g. we do not put important information at the ends of the sequence).